ImpDAR Documentation

Release 1.0.0

David Lilien

Apr 16, 2020

Contents:

ImpDAR is a flexible, open-source impulse radar processor that provides most of the benefits (and some additional features) compared to expensive commercial software. The starting point was the old St. Olaf deep radar matlab code. This code has a lot of history of contributors–I've tried to preserve acknowledgment of many of them in the file headers.

Support is gradually being added for a variety of file formats. Currently, GSSI, PulseEKKO, Radan, Blue Systems, DELORES, SEGY, gprMAX, seidart, Gecko, and legacy StoDeep files are supported. Available processing steps include various filtering operations, trivial modifications such as restacking, cropping, or reversing data, and a few different geolocation-related operations like interpolating to constant trace spacing. The primary interface is through the command line, which allows efficient processing of large volumes of data. An API, centered around the RadarData class, is also available to allow the user to use ImpDAR in other programs.

In addition to processing, ImpDAR can also be used for picking reflectors. Picking is generally an interactive process, and there is a light GUI for doing the picking. The GUI also provides support for basic processing operations, so you can see the effect of steps as you go along.

CHAPTER 1

Requirements

Python 2.7+ or 3.4+

numpy, scipy, matplotlib

To do anything involving geolocation, you will also need GDAL. The GUI, which is needed to be able to pick reflectors, requires PyQt5. SegYIO is needed for SEGY support and for SeisUnix migration. h5py is needed for some data formats.

CHAPTER 2

Installation

2.1 Beginner

If you do not have a current (2.7 or 3+) python installation, you will need one to begin. I recommend getting python 3 from anaconda. The Anaconda installer is straightforward to use, and you can let it set up your path, which makes the subsequent commands "just work." However, Anaconda on Windows suggests not putting it on your path and instead using the Anaconda prompt. The procedure is the same–just open an anaconda prompt window after installation then continue. If you are on MacOS or Linux, you will want to restart your terminal after installing Anaconda so you get updated path specs.

Next, we need to install dependencies. GDAL is needed for accurate measurement of distance, and for converting coordinate systems. I recommend getting it and segyio, used for interacting with the SEGY data format, using,

```
conda install -c conda-forge gdal segyio
```

This step can be really slow, so don not worry if it is a bit painful. At this point, I also recommend installing h5py, just so that you are ready for all presently supported formats. This can be done with

```
conda install h5py
```

Now, you are ready to install impdar. You can get a version with

```
pip install impdar
```

If you are not a super user, you may get an error related to permissions. This is fine, you just need to install for yourself only. Use

```
pip install --user impdar
```

You should now be all set to start using ImpDAR. Scroll down for documentation and links for examples.

2.2 Advanced

If you are not using Anaconda, you are on your own for installing dependencies. The challenges are generally GDAL and PyQt5, since these rely on libraries in other languages. For the most basic use cases, you can skip these, and go straight to installing ImpDAR with pip or through github.

To be sure that you have the newest version of ImpDAR as a lot of development is happening, you will want to use the development branch from GitHub. The pypi (pip) version is not updated as often to ensure a stable release. To get the devel version off git,

```
git clone -b devel https://github.com/dlilien/ImpDAR.git
cd impdar
python setup.py install
```

This requires git.

If you want to have the full suite of migration options, you will need to install seisunix. The SeisUnix install is bit complicated, but there are instructions with it. It should be possible to use SeisUnix on Windows with CygWin then interface with ImpDAR, but this is untested.

CHAPTER 3

Examples

Check out the *examples*, particularly the Jupyter notebook examples beginning with *getting started*, for an idea of how to run ImpDAR. These should be a good starting point that can be modified for a particular use case. While all of the output and input are on this website, if you actually want to run the code you can download all the notebooks and run them yourself. You can get those here.

CHAPTER 4

Contributing

I would be thrilled to get pull requests for any additional functionality. In particular, it is difficult for me to add support for input formats for which I do not have example data–any development of readers for additional data types would be greatly appreciated.

4.1 Installation

4.1.1 Beginner

If you do not have a current (2.7 or 3+) python installation, you will need one to begin. I recommend getting python 3 from anaconda. The Anaconda installer is straightforward to use, and you can let it set up your path, which makes the subsequent commands "just work." However, Anaconda on Windows suggests not putting it on your path and instead using the Anaconda prompt. The procedure is the same–just open an anaconda prompt window after installation then continue. If you are on MacOS or Linux, you will want to restart your terminal after installing Anaconda so you get updated path specs.

Next, we need to install dependencies. GDAL is needed for accurate measurement of distance, and for converting coordinate systems. I recommend getting it and segyio, used for interacting with the SEGY data format, using,

```
conda install -c conda-forge gdal segyio
```

This step can be really slow, so don not worry if it is a bit painful. At this point, I also recommend installing h5py, just so that you are ready for all presently supported formats. This can be done with

```
conda install h5py
```

Now, you are ready to install impdar. You can get a version with

```
pip install impdar
```

If you are not a super user, you may get an error related to permissions. This is fine, you just need to install for yourself only. Use

```
pip install --user impdar
```

You should now be all set to start using ImpDAR. Scroll down for documentation and links for examples.

4.1.2 Advanced

If you are not using Anaconda, you are on your own for installing dependencies. The challenges are generally GDAL and PyQt5, since these rely on libraries in other languages. For the most basic use cases, you can skip these, and go straight to installing ImpDAR with pip or through github.

To be sure that you have the newest version of ImpDAR as a lot of development is happening, you will want to use the development branch from GitHub. The pypi (pip) version is not updated as often to ensure a stable release. To get the devel version off git,

```
git clone -b devel https://github.com/dlilien/ImpDAR.git
cd impdar
python setup.py install
```

This requires git.

If you want to have the full suite of migration options, you will need to install seisunix. The SeisUnix install is bit complicated, but there are instructions with it. It should be possible to use SeisUnix on Windows with CygWin then interface with ImpDAR, but this is untested.

4.2 API

This section documents the classes and functions of the libraries underlying ImpDAR. These really are the workhorses behind the executables that you would use for command-line processing. On the other hand, if you want to integrate the processing steps implemented by ImpDAR into another program, you will be interacting with these libraries.

The central component of ImpDAR processing is the `RadarData` class. Not only does this object store all the radar returns and auxiliary information, it also has a number of methods for processing.

Some processing steps may be implemented separately from the `RadarData` class. At present, just `concatenation`, is separate because it acts on multiple `RadarData` objects.

Contents:

4.2.1 RadarData

This page contains the documentation for the RadarData class, which is the basic object in ImpDAR. If you are interacting with the API in a significant way, this is where you will find documentation from most of the things you care about, particularly how the data is stored and how to do basic processing steps on it. All of the files to define the class are in impdar/lib/Radardata, with the basic initialization and class properties found in __init__.py and addional functionality spread across _RadarDataSaving, _RadarDataFiltering, and _RadarDataProcessing.

RadarData Base

class impdar.lib.RadarData.**RadarData**(*fn_mat*)

A class that holds the relevant information for a radar profile.

We keep track of processing steps with the flags attribute. This base version's __init__ takes a filename of a .mat file in the old StODeep format to load.

attrs_guaranteed = ['chan', 'data', 'decday', 'dt', 'pressure', 'snum', 'tnum', 'trace_
Attributes that every RadarData object should have. These should not be None.

attrs_optional = ['nmo_depth', 'lat', 'long', 'elev', 'dist', 'x_coord', 'y_coord', 'f
Optional attributes that may be None without affecting processing. These may not have existed in old StoDeep files, and they often cannot be set at the initial data load. If they exist, they all have units of meters.

chan = None
The Channel number of the data.

check_attrs()
Check if required attributes exist.

This is largely for development only; loaders should generally call this method last, so that they can confirm that they have defined the necessary attributes.

Raises `ImpdarError` – If any required attribute is None, or any optional attribute is fully absent

data = None
np.ndarray(snum x tnum) of the actual return power.

decday = None
np.ndarray(tnum,) of the acquisition time of each trace note that this is referenced to Jan 1, 0 CE (matlabe datenum) for convenience, use the *datetime* attribute to access a python version of the day

dist = None
np.ndarray(tnum,) of the distances along the profile. units will depend on whether geographic coordinate transforms, as well as GPS data, are available.

dt = None
float, The spacing between samples in travel time, in seconds.

elev = None
np.ndarray(tnum,) Optional. Elevation along the profile.

fn = None
str, the input filename. May be left as None.

lat = None
np.ndarray(tnum,) latitude along the profile. Generally not in projected coordinates

long = None
np.ndarray(tnum,) longitude along the profile. Generally not in projected coords.

nmo_depth = None
np.ndarray(tnum,) Optional. Depth of each trace below the surface

pressure = None
np.ndarray(tnum,) The pressure at acquisition. ImpDAR does not use this at present.

snum = None
int number of samples per trace

tnum = None
int, the number of traces in the file

trace_int = None
float, the time between traces.

trace_num = None
np.ndarray(tnum,) The 1-indexed number of the trace

travel_time = None
np.ndarray(snum,) The two way travel time to each sample, in us

trig = None
np.ndarray(tnum,) the index in each trace where the triggered.

trig_level = None
float, The value on which the radar was triggering.

x_coord = None
np.ndarray(tnum,) Optional. Projected x-coordinate along the profile.

y_coord = None
np.ndarray(tnum,) Optional. Projected y-coordinate along the profile.

Saving RadarData

These are all instance methods for saving information from a RadarData object. They are defined in impdar/lib/RadarData/_RadarDataSaving.py.

`RadarData.`**save**(*fn*)
Save the radar data

Parameters **fn** (*str*) – Filename. Should have a .mat extension

`RadarData.`**save_as_segy**(*fn*)

`RadarData.`**output_shp**(*fn*, *t_srs=4326*, *target_out=None*)
Output a shapefile of the traces.

If there are any picks, we want to output these. If not, we will only output the tracenumber. This function requires osr/gdal for shapefile creation. I suggest exporting a csv if you don't want to deal with gdal.

Parameters

- **fn** (*str*) – The filename of the output
- **t_srs** (*int, optional*) – EPSG number of the target spatial reference system. Default 4326 (wgs84)
- **target_out** (*str, optional*) – Used to overwrite the default output format of picks. By default, try to write depth and if there is no nmo_depth use TWTT. You might want to use this to get the output in TWTT or sample number (options are depth, elev, twtt, snum)

`RadarData.`**output_csv**(*fn*, *target_out=None*, *delimiter=', '*)
Output a csv of the traces.

If there are any picks, we want to output these. If not, we will only output the tracenumber.

Parameters

- **fn** (*str*) – The filename of the output
- **target_out** (*str, optional*) – Used to overwrite the default output format of picks. By default, try to write depth and if there is no nmo_depth use TWTT. You might want to use this to get the output in TWTT or sample number (options are depth, elev, twtt, snum)
- **delimiter** (*str, optional*) – Delimiter for csv. Default ','.

Processing RadarData

These are all instance methods for processing data on a RadarData object. They are defined in impdar/lib/RadarData/_RadarDataProcessing.py.

`RadarData.`**`reverse`**`()`

Reverse radar data

Essentially flip the profile left-right. This is desirable in a number of instances, but is particularly useful for concatenating profiles acquired in opposite directions. The St Olaf version of this function had a bunch of options. They seemed irrelevant to me. We just try to flip everything that might need flipping.

`RadarData.`**`nmo`**`(`*ant_sep*, *uice=169000000.0*, *uair=300000000.0*, *rho_profile=None*, *permittivity_model=<function firn_permittivity>*, *const_sample=True*`)`

Normal move-out correction.

Converts travel time to distance accounting for antenna separation. This potentially affects data and snum. It also defines nmo_depth, the moveout-corrected depth

Updated to accomodate vertical velocity profile. B Hills 8/2019

Parameters

- **`ant_sep`** (*`float`*) – Antenna separation in meters.
- **`uice`** (*`float or np.ndarray (2 x m), optional`*) – Speed of light in ice, in m/s. (different from StoDeep!!). Default 1.69e8.
- **`uair`** (*`float, optional`*) – Speed of light in air. Default 3.0e8
- **`rho_profile`** (*`str,optional`*) – Filename for a csv file with density profile (depths in first column and densities in second) Units should be meters for depth, kgs per meter cubed for density. Note that using a variable uice will break the linear scaling between depth and time, so we are forced to choose whether the y-axis is linear in speed or time. I chose time, since this eases calculations like migration. For plotting vs. depth, however, the functions just use the bounds, so the depth variations are averaged out. You can use the helper function constant_sample_depth_spacing() in order to correct this, but you should call that after migration.
- **`permittivity_model`** (*`fun, optional`*) – density to permittivity model from the literature
- **`const_sample`** (*`bool, optional`*) – interpolate to constant sample spacing after the nmo correction

`RadarData.`**`crop`**`(`*lim*, *top_or_bottom='top'*, *dimension='snum'*, *uice=169000000.0*, *rezero=True*, *zero_trig=True*`)`

Crop the radar data in the vertical. We can take off the top or bottom.

This will affect data, travel_time, and snum.

Parameters

- **`lim`** (*`float (int if dimension=='snum')`*) – The value at which to crop.
- **`top_or_bottom`** (*`str, optional`*) – Crop off the top (lim is the first remaining return) or the bottom (lim is the last remaining return).
- **`dimension`** (*`str, optional`*) – Evaluate in terms of sample (snum), travel time (twtt), or depth (depth). If depth, uses nmo_depth if present and use uice with no transmit/receive separation. If pretrig, uses the recorded trigger sample to crop.
- **`rezero`** (*`bool, optional`*) – Set the zero on the y axis to the cropped value (if cropping off the top). Default True. This is desirable if you are zeroing to the surface.

- **zero_trig** (*bool, optional*) – Reset the trigger to zero. Effectively asserts that the crop was to the surface. Default True.
- **uice** (*float, optional*) – Speed of light in ice. Used if nmo_depth is None and dimension=='depth'

RadarData.**restack**(*traces*)

Restack all relevant data to the given number of traces.

This function just takes the average of the given number of traces. This reduces file size and can get rid of noise. There are fancier ways to do this— if you have GPS, you probably want to restack to constant trace spacing instead.

Parameters traces (*int*) – The (odd) number of traces to stack

RadarData.**rangegain**(*slope*)

Apply a range gain.

Parameters slope (*float*) – The slope of the linear range gain to be applied. Maybe try 1.0e-2?

RadarData.**agc**(*window=50, scaling_factor=50*)

Try to do some automatic gain control

This is from StoDeep–I'm not sure it is useful but it was easy to roll over so I'm going to keep it. I think you should have most of this gone with a bandpass, but whatever.

Parameters

- **window** (*int, optional*) – The size of window we use in number of samples (default 50)
- **scaling_factor** (*int, optional*) – The scaling factor. This gets divided by the max amplitude when we rescale the input. Default 50.

RadarData.**constant_space**(*spacing, min_movement=0.01, show_nomove=False*)

Restack the radar data to a constant spacing.

This method uses the GPS information (i.e. the distance, x, y, lat, and lon), to do a 1-d interpolation to get new values in the radargram. It also updates related variables like lat, long, elevation, and coordinates. To avoid retaining sections of the radargram when the antenna was in fact stationary, some minimum movement between traces is enforced. This value is in meters, and should change to be commensurate with the collection strategy (e.g. skiing a radar is slower than towing it with a snowmobile).

This function comprises the second half of what was done by StoDeep's interpdeep. If you have GPS data from an external, high-precision GPS, you would first want to call *impdar.lib.gpslib.kinematic_gps_control* so that the GPS-related variables are all improved, then you would want to call this method. *impdar.lib.gpslib* provides some wrappings for doing both steps and for loading in the external GPS data.

Parameters

- **spacing** (*float*) – Target trace spacing, in meters
- **min_movement** (*float, optional*) – Minimum trace spacing. If there is not this much separation, toss the next shot. Set high to keep everything. Default 1.0e-2.
- **show_nomove** (*bool, optional*) – If True, make a plot shading the areas where we think there is no movement. This can be really helpful for diagnosing what is wrong if you have lingering stationary traces.

RadarData.**elev_correct**(*v_avg=169000000.0*)

Move the surface down in the data array to account for surface elevation.

NMO depth attribute must have been created before elev_correct is called. This method should generally be called after you have interpolated precision GPS onto the data, otherwise the noise a handheld GPS will make the results look pretty bad.

Be aware that this is not a precise conversion if your nmo correction had antenna separation, or if you used a depth-variable velocity structure. This is because we have a single vector describing depths in the data array, so only one value for each depth step.

Parameters **`v_avg`** (`float, optional`) – Average velocity. This is what will define the depth slices in the new data array. Default 1.69e8.

Raises ValueError: – If there is no nmo_depth since this is used for calculating depths

Filtering Radar Data

These are all instance methods for filtering data to remove noise. They are defined in impdar/lib/RadarData/_RadarDataFiltering.py.

`RadarData.`**`migrate`**(*mtype='stolt'*, *vtaper=10*, *htaper=10*, *tmig=0*, *vel_fn=None*, *vel=168000000.0*, *nxpad=10*, *nearfield=False*, *verbose=0*)

Migrate the data.

This is a wrapper around all the migration routines in migration_routines.py.

Parameters **`mtype`** (`str, optional`) – The chosen migration routine. Options are: kirch, stolt, phsh. Default: stolt

`RadarData.`**`vertical_band_pass`**(*low*, *high*, *order=5*, *filttype='butter'*, *cheb_rp=5*, *fir_window='hamming'*, **args*, ***kwargs*)

Vertically bandpass the data

This function uses a forward-backward filter on the data. Returns power that is not near the wavelength of the transmitter is assumed to be noise, so the limits for the filtering should generally surround the radar frequency. Some experimentation may be needed to see what gives the clearest results for any given data

There are a number of options for the filter type. Depending on the type of filter chosen, some other p

Parameters

- **`low`** (`float`) – Lowest frequency passed, in MHz
- **`high`** (`float`) – Highest frequency passed, in MHz
- **`order`** (`int`) – Filter order (default 5)
- **`filttype`** (`str, optional`) – The filter type to use. Options are butter(worth), cheb(yshev type I), bessel, or FIR (finite impulse response). Default is butter.
- **`cheb_rp`** (`float, optional`) – Maximum ripple, in decibels, of Chebyshev filter. Only used if filttype=='cheb'. Default is 5.
- **`fir_window`** (`str, optional`) – The window type passed to scipy.signal.firwin. Only used if filttype=='fir'. Default is hamming'

`RadarData.`**`adaptivehfilt`**(*window_size=1000*, **args*, ***kwargs*)

Adaptively filter to reduce noise in upper layers

This subtracts the average of traces around an individual trace in order to filter it. You can call this method directly, or it can be called by sending the 'adaptive' option to `RadarData.hfilt()`

Original StoDeep Documentation: HFILTDEEP-This StoDeep subroutine processes bandpass filtered or NMO data to reduce the horizontal noise in the upper layers. The user need not specify any frequencies. This program simply takes the average of all of the traces and subtracts it from the bandpassed data.

It will remove most horizontally-oriented features in a radar profile: ringing, horizontal reflectors. It will also create artifacts at travel-times corresponding to any bright horizontal reflectors included in the average trace.

You will want to experiment with the creation of the average trace. Ideally choose an area where all reflectors are sloped and relatively dim so that they average out while the horizontal noise is amplified. Note that generally there is no perfect horizontal filter that will work at all depths. You will have to experiment to get the best results for your area of interest.

WARNING: Do not use hfiltdeep on elevation-corrected data!!!

Created: Logan Smith - 6/12/02 Modified by: L. Smith, 5/27/03. K. Dwyer, 6/3/03. B. Welch, 5/2/06. B. Youngblood, 6/13/08. J. Olson, 7/10/08

`RadarData.`**`horizontalfilt`**(*ntr1*, *ntr2*, **args*, ***kwargs*)

Remove the average trace.

Parameters

- **`ntr1`** (*`int`*) – Leftmost trace for averaging
- **`ntr2`** (*`int`*) – Rightmost trace for averaging

Original StoDeep Documentation: HFILTDEEP - This StoDeep subroutine processes bandpass filtered or NMO data to reduce the horizontal noise in the upper layers. The user need not specify any frequencies. This program simply takes the average of all of the traces and subtracts it from the bandpassed data. It will remove most horizontally-oriented features in a radar profile: ringing, horizontal reflectors. It will also create artifacts at travel-times corresponding to any bright horizontal reflectors included in the average trace.

You will want to experiment with the creation of the average trace. Ideally choose an area where all reflectors are sloped and relatively dim so that they average out while the horizontal noise is amplified. Note that generally there is no perfect horizontal filter that will work at all depths. You will have to experiment to get the best results for your area of interest.

`RadarData.`**`highpass`**(*wavelength*)

High pass in the horizontal for a given wavelength.

This only works if the data have constant trace spacing; we check the processing flags to enforce this.

Parameters **`wavelength`** (*`int`*) – The wavelength to pass, in meters.

Original StoDeep Documentation: HIGHPASSDEEP - This is NOT a highpass frequency filter–rather it is a horizontal filter to be used after interpolation because our data now has constant spacing and a constant time. Note that this horizontal filter requires constant trace-spacing in order to be effective.

You will want to experiment with the creation of the average trace. Ideally choose an area where all reflectors are sloped and relatively dim so that they average out while the horizontal noise is amplified. Note that generally there is no perfect horizontal filter that will work at all depths. You will have to experiment to get the best results for your area of interest.

WARNING: Do not use highpassdeep on elevation-corrected data!!!

Created by L. Smith and modified by A. Hagen, 6/15/04. B. Welch, 5/3/06. J. Werner, 6/30/08. J. Olson, 7/10/08

`RadarData.`**`winavg_hfilt`**(*avg_win*, *taper='full'*, *filtdepth=100*)

Uses a moving window to find the average trace, then subtracts this from the data.

Parameters

- **avg_win** (*int*) – The size of moving window. Must be odd and less than tnum. We will correct these rather than raise an exception.
- **taper** (*str*) – How the filter varies with depth. Options are full or pexp. For full, the power tapers exponentially. For pexp, the filter stops after the sample number given by filtdepth.

Original StoDeep Documentation:

WINAVG_HFILTDEEP - This StoDeep subroutine performs a horizontal filter on the data to reduce the ringing in the upper layers. It uses a moving window to create an average trace for each individual trace, applies an exponential taper to it and then subtracts it from the trace. This is based off of the original hfiltdeep and uses the moving window designed in cosine_win_hfiltdeep. The extent of the taper can help to minimize artifacts that are often produced near regions of bright bed reflectors.

You will want to experiment with the creation of the average trace. Ideally choose an area where all reflectors are sloped and relatively dim so that they average out while the horizontal noise is amplified. Note that generally there is no perfect horizontal filter that will work at all depths. You will have to experiment to get the best results for your area of interest.

WARNING: Do not use winavg_hfiltdeep on elevation-corrected data!!!

Created: Kieran Dwyer 6/18/03 Modified by B. Welch, 5/2/06. J. Olson, 7/10/08.

`RadarData.`**`hfilt`** (*ftype='hfilt'*, *bounds=None*)

Horizontally filter the data.

This is a wrapper around other filter types. Horizontal filters are implemented (and documented) in the `impdar.lib.horizontal_filters` module.

Parameters

- **ftype** (*str, optional*) – The filter type. Options are `hfilt` and `adaptive`. Default hfilt
- **bounds** (*tuple, optional*) – Bounds for the hfilt. Default is None, but required if ftype is hfilt.

4.2.2 Plotting

Plotting functions for radar data.

`impdar.lib.plot.`**`plot`** (*fns*, *tr=None*, *s=False*, *ftype='png'*, *dpi=300*, *xd=False*, *yd=False*, *x_range=(0, -1)*, *power=None*, *spectra=None*, *freq_limit=None*, *window=None*, *scaling='spectrum'*, *filetype='mat'*, *pick_colors=None*, *ft=False*, *hft=False*, *clims=None*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, *flatten_layer=None*, **args*, ***kwargs*)

Wrap a number of plot types.

This should really only be used by the exectuables. If you are plotting yourself, just use the individual plotting functions that are described below.

Parameters

- **fns** (*list of strs*) – A list of filenames to plot individually.
- **tr** (*tuple or int, optional*) – Plot traces tr[1] to tr[2] (or trace tr) rather than the radargram. Default is None (plot radargram).
- **power** (*int, optional*) – If not None, then plot power returned from this layer

- **`filetype`** (*`str, optional`*) – Type of input file. Default mat.
- **`x_range`** (*`tuple, optional`*) – The range of traces to plot in the radargram. Default is (0, -1) (plot all traces)
- **`flatten_layer`** (*`int, optional`*) – Distort the radargram so this layer is flat. Default is None (do not distort).

`impdar.lib.plot.`**`plot_ft`**(*dat*, *fig=None*, *ax=None*, ***line_kwargs*)

Plot the Fourier spectrum of the data in the vertical.

This will give the power spectral density in terms of the frequency (in MHz). We first fft, then average the fft.

Parameters

- **`dat`** (*`impdar.lib.RadarData.Radardata`*) – The RadarData object to plot.
- **`fig`** (*`matplotlib.pyplot.Figure`*) – Figure canvas that should be plotted upon
- **`ax`** (*`matplotlib.pyplot.Axes`*) – Axes that should be plotted upon
- **`**line_kwargs`** – Arguments passed to the plotting call (e.g. color, linewidth)

Returns

- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that was plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that were plotted upon

`impdar.lib.plot.`**`plot_hft`**(*dat*, *fig=None*, *ax=None*)

Plot the Fourier spectrum of the data in the horizontal.

This will give the power spectral density as a function of the horizontal wavelength (in meters). We first fft, then average the fft

Parameters

- **`dat`** (*`impdar.lib.RadarData.Radardata`*) – The RadarData object to plot.
- **`fig`** (*`matplotlib.pyplot.Figure`*) – Figure canvas that should be plotted upon
- **`ax`** (*`matplotlib.pyplot.Axes`*) – Axes that should be plotted upon

Returns

- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that was plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that were plotted upon

`impdar.lib.plot.`**`plot_picks`**(*rd*, *xd*, *yd*, *colors=None*, *flatten_layer=None*, *fig=None*, *ax=None*)

Update the plotting of the current pick.

Parameters

- **`colors`** (*`str`*) – You have choices here. This can be a npicksx3 list, an npicks list of 3-letter strings, a 3 letter string, a single string, or a npicks list. Any of the x3 options are interpretted as top, middle, bottom colors. If it is a string, the lines are all plotted in this color. If it is a list, the different values are used for the different lines.
- **`flatten_layer`** (*`int, optional`*) – Make this layer flat in the plot. Distorts all layers. Default is no distortion.

`impdar.lib.plot.`**`plot_power`**(*dats*, *idx*, *fig=None*, *ax=None*, *clims=None*)

Make a plot of the reflected power along a given pick.

Parameters

- **`dat`** (*`impdar.lib.RadarData.Radardata`*) – The RadarData object to plot.

- **idx** (*int*) – A picknum in the dat.picks.picknum array
- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that should be plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that should be plotted upon

Returns

- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that was plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that were plotted upon

`impdar.lib.plot.`**`plot_radargram`**(*dat*, *xdat='tnum'*, *ydat='twtt'*, *x_range=(0, -1)*, *y_range=(0, -1)*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, *fig=None*, *ax=None*, *return_plotinfo=False*, *pick_colors=None*, *clims=None*, *flatten_layer=None*)

Plot a radio echogram.

This function is a little weird since I want to be able to plot on top of existing figures/axes or on new figures an axes. There is therefore an argument *return_plotinfo* that funnels between these options and changes the return types.

Parameters

- **dat** (*impdar.lib.RadarData.Radardata*) – The RadarData object to plot.
- **xdat** (*str, optional*) – The horizontal axis units. Either tnum or dist(ance).
- **ydat** (*str, optional*) – The vertical axis units. Either twtt or or depth. Default twtt.
- **x_range** (*2-tuple, optional*) – The range of values to plot, in tnum space. Default is plot everything (0, -1)
- **y_range** (*2-tuple, optional*) – The range of values to plot, in snum space. Default is plot everything (0, -1)
- **cmap** (*matplotlib.pyplot.cm, optional*) – The colormap to use
- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that should be plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that should be plotted upon

Returns

- *If not return_plotinfo* –

  fig: matplotlib.pyplot.Figure Figure canvas that was plotted upon

  ax: matplotlib.pyplot.Axes Axes that were plotted upon

- *else* –

  im: pyplot.imshow The image object plotted

  xd: np.ndarray The x values of the plot

  yd: np.ndarray The y values of the plot

  x_range: 2-tuple The limits of the x range, after modification to remove negative indices

  clims: 2-tuple The limits of the colorbar

`impdar.lib.plot.`**`plot_spectrogram`**(*dat*, *freq_limit=None*, *window=None*, *scaling='spectrum'*, *fig=None*, *ax=None*, ***kwargs*)

Make a plot of power spectral density across all traces of a radar profile.

Parameters

- **dat** (*impdar.lib.RadarData.Radardata*) – The RadarData object to plot.

- **freq_limit** (*tuple*) – The minimum and maximum frequency (in MHz) to limit the y-axis to
- **window** (*str, optional*) – Type of window to be used for the signal.periodogram() method.

  Default hamming. Further information
- **scaling** (*str, optional*) – Whether to plot power spectral density or power spectrum 'density' or 'spectrum', the default being 'spectrum'. Further information
- **fig** (*matplotlib.pyplot.Figure, optional*) – Figure canvas that should be plotted upon
- **ax** (*matplotlib.pyplot.Axes, optional*) – Axes that should be plotted upon

Returns

- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that was plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that were plotted upon

`impdar.lib.plot.`**`plot_traces`**(*dat, tr, ydat='twtt', fig=None, ax=None, linewidth=1.0, linestyle='solid'*)

Plot power vs depth or twtt in a trace.

Parameters

- **dat** (*impdar.lib.RadarData.Radardata*) – The RadarData object to plot.
- **tr** (*int or 2-tuple*) – Either a single trace or a range of traces to plot
- **ydat** (*str, optional*) – The vertical axis units. Either twtt or or depth. Default twtt.
- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that should be plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that should be plotted upon

Returns

- **fig** (*matplotlib.pyplot.Figure*) – Figure canvas that was plotted upon
- **ax** (*matplotlib.pyplot.Axes*) – Axes that were plotted upon

4.2.3 Interpretation

Interpretation in this context primarily means picking layers (either isochrones or the bed). In the future, this functionality may be expanded to make picking other things, e.g. some discontinuity, easier.

Functions used for picking

Functions that are a for the mechanics of picking, not for the display.

`impdar.lib.picklib.`**`get_intersection`**(*data_main, data_cross, return_nans=False*)

Find the intersection of two radar datasets.

Used for plotting up where pick depths at places where two profiles cross. This is a pretty simple function as implemented, so it is not going to find you multiple intersections. Rather, you are just going to get the single point where the traces are closest. Note that this will work picking sequential profiles too. If there are nans at the intersection, we look for the closest non-nan.

Parameters

- **`data_main`** (`impdar.lib.RadarData.RadarData`) – The RadarData object upon which you want the intersection in reference to (i.e. then one you will plot up)
- **`data_cross`** (`impdar.lib.RadarData.RadarData`) – The crossprofile from which you are going to plot up layers. Must have some picks in it.
- **`return_nans`** (*`bool, optional`*) – Return the closest sample, even if it was nan-picked. Default is false (find closest non-nan value)

Returns

- *np.ndarray (npicks,)* – The tracenumber in the main profile at which we are getting the sample from the crossprofile
- *np.ndarray (npicks,)* – The depths (in sample number) of the layers in the cross profile. Note that this essentially assume you are using the same snum/depth conversion between the two profiles.

Raises AttributeError: – if there are no picks in the cross profile.

`impdar.lib.picklib.`**`packet_pick`**(*trace*, *pickparams*, *midpoint*)

Really do the picking.

This is where we look for the highest amplitude return, and the opposite polarity returns surrounding it

Parameters

- **`trace`** (*`1d numpy.ndarray`*) – The trace in which we are looking for our return
- **`pickparams`** (`impdar.lib.PickParameters.PickParameters`) – The information about picking that we need for determining window size and polarity
- **`midpoint`** (*`int`*) – The guess at the index where a pick is, used for searching out the highest amplitude return

Returns len=5. Top of packet, middle of packet, bottom of packet, nan, power

Return type list

`impdar.lib.picklib.`**`packet_power`**(*trace*, *plength*, *midpoint*)

Return power of a packet.

This function is pretty boring. It just finds a window around a point in a trace.

Parameters

- **`trace`** (*`numpy.ndarray`*) – (snum,) The trace in which to find the window
- **`plength`** (*`float`*) – The size of the packet (in samples)
- **`midpoint`** (*`int`*) – The central sample

Returns

- *numpy.ndarray* – The packet of length plength
- *int* – The index of the top sample (desirable for calculating overall indices in functions that call this one).

`impdar.lib.picklib.`**`pick`**(*traces*, *snum_start*, *snum_end*, *pickparams*)

Pick a reflector in some traces.

Uses a line between the starting and ending picks to guide picking the maximum (or minimum) return, and the surrounding peaks with opposite polarity.

Parameters

- **traces** (*numpy.ndarray*) – The chunk of data we are picking. Should be a slice of the overall data with shape snum x (size to pick)
- **snum_start** (*int*) – The index of the pick in the leftmost trace. We would normally get this from the last pick.
- **snum_end** (*int*) – The index of the pick in the rightmost trace.
- **pickparams** (*impdar.lib.PickParameters.PickParameters*) – Use for polarity, frequency, plength.

Returns The picks selected. Rows are: top of packet, center pick, bottom of packet, time (deprecated, all nans), and power. Size 5xtnum

Return type numpy.ndarray

Classes used by interpreter

These classes are broken down to match the structure of StODeep, so we store information about how the picks get made, and the picks themselves, using different objects.

If you have done some interpretation, you will likely want to subsequently interact with the *Picks* object. Often, this can be done without accessing the API by converting the picks/geospatial data to another form, e.g. with *impdar convert shp fn_picked.mat*. You can also make plots with the picks on top of the radar data, or with the return power in geospatial coordinates, using *impplot rg fn_picked.mat* or *impplot power fn_picked.mat layer_num*. For further operations, you will probably want to access the *Picks* object described next. For example, using the picks object you could do something like

```
import numy as np
import matplotlib.pyplot as plt
from impdar.lib import RadarData
rd = RadarData('[PICKED_DATA_FN.mat]')

# make a basic plot of the radargram
fig, ax = plt.subplots()
im, _, _, _, _ = plot.plot_radargram(rd, fig=fig, ax=ax, xdat='dist', ydat='depth',␣
↪return_plotinfo=True)

# calculate the return power
c = 10. * np.log10(rd.picks.power[0, :])
c -= np.nanmax(c)

# plot the return power on the layer, being careful of NaNs
mask = ~np.isnan(rd.picks.samp1[0, :])
cm = ax.scatter(rd.dist[mask.flatten()],
                rd.nmo_depth[rd.picks.samp1[0, :].astype(int)[mask]],
                c=c.flatten()[mask.flatten()],
                s=1)
```

The Picks structure tracks picks and picking parameters.

class impdar.lib.Picks.**Picks**(*radardata*, *pick_struct=None*)

Information about picks.

This object holds all picks for a given radargram. The main containers are matrices holding information about the indices in the data matrix a given pick lies.

samp1

Min/max above the center of each pick. A new row for each new pick.

Type nsamp x tnum array

samp2
Max/min at the center of each pick. A new row for each new pick.

Type nsamp x tnum array

samp3
Max/min below the center of each pick. A new row for each new pick.

Type nsamp x tnum array

time
In StoDeep used to contain TWTT samp2. Since this is redundant I'm deptrecating it, and it will be zeros for impdar processed data.

Type nsamp x tnum array

power
Power across a pick. To get this in decibels, you need to take 10. * np.log10(power)

Type nsamp x tnum array

picknums
The number of each pick.

Type list of length nsamp

lasttrace
Information about the end of the trace

Type impdar.lib.LastTrace.LastTrace

lt
StoDeep legacy for compatibility, unused

Type impdar.lib.LeaderTrailer.LeaderTrailer

pickparams
This structure contains important information used in picking, such as frequency for picks.

Type *impdar.lib.PickParameters.PickParameters*

add_pick(*picknum=0*)
Add a new pick.

This method handles any complexity in adding a new pick. If no picks exist, it creates the matrices. Otherwise, it just adds rows. If the last pick hasn't been used, this just recycles that pick.

Parameters **picknum** (*int, optional*) – Number to call the new pick. Default zero

Returns **index** – The index of the row number containing the new pick

Return type int

Raises ValueError if the picknum already exists–we do not deal with repeats

to_struct()
Convert to a format writable to a .mat file.

Returns **mat** – Dictionary of attributes for export with scipy.io.savemat

Return type dict

update_pick(*picknum*, *pick_info*)
Update a pick with new information.

Rather than go in and manually update a pick every time it is changed, we take in all information about an individual pick simultaneously and this method updates the pick's information in the Pick object.

Parameters

- **picknum** (*int*) – The pick number to update. Must be in picknums
- **pick_info** (*5xtnum np.ndarray*) – Array where rows are upper pick bound, pick center, lower pick bound, time (deprecated, usually left as zeros or nans), and power across the pick.

Raises

- ValueError if picknum is not in picknums or if the shape of the
- pick_info is bad.

Structure with input data for the picking algoriths.

class `impdar.lib.PickParameters.`**`PickParameters`**(*radardata*, *pickparams_struct=None*)

Some information used for determining for picks.

This object contains several things that you need to know in order to pick a radar layer, like the frequency of layers you are looking for and the size window in which to search.

apickthresh

Some kind of auto picking threshold (Unused: default 10)

Type float

freq

Frequency of the layer pick (default 4)

Type float

dt

Time between acquisitions

Type float

plength

The total packet to search for peaks

Type float

FWW

The width of the center portion which we are going to search

Type float

scst

The offset which we will search at

Type float

pol

Polarity of the picks

Type int

apickflag

I think this just kept track of whether StoDeep was autopicking

Type int

addpicktype

Some flag

Type str

radardata
A link back up to the RadarData object with which this is affiliated

Type *RadarData*

freq_update (*freq*)
Update the frequency at which we are looking.

This is more complicated than just setting freq because other variables are a function of frequency and if not updated will break.

Parameters **freq** (`float`) – Target pick frequency.

to_struct ()
Return attributes as a dictionary for saving.

Guards against Nones so we can export to matlab

Returns The data for export with scipy.io.savemat

Return type dict

4.2.4 Loading data

These are functions for loading loading radar data, generally from raw formats, to be used in a program or saved in ImpDAR's .mat format and used later.

For every filetype that ImpDAR can handle (e.g. GSSI .DZT files, gprMax .h5 files), there is a dedicated file for loading that filetype in *impdar/lib/load*. These files generally define a single method, which returns an *impdar.lib.RadarData.RadarData* object, with information specific to the filetype loaded in. The user does not need to interact with these files (unless they need to add functionality).

Instead, to load data for interactive use, a generic *load* command, which takes a filetype as an argument, is defined in *impdar.lib.load.__init__*. This wrapper provides some conveniences for handling multiple files as well. There is also a *load_and_exit* command in that file, which can be used if the user does not want to interact with the data at load time, but wants the filetype converted to ImpDAR's .mat for convenience.

`load.`**load** (*fns_in*, *channel=1*, **args*, ***kwargs*)
Load a list of files of a certain type

Parameters

- **filetype** (`str`) – The type of file to load.
- **fns** (`list`) – List of files to load
- **channel** (`Receiver channel that the data were recorded on`) – This is primarily for the St. Olaf HF data

Returns **RadarDataList** – Objects with relevant radar information

Return type list of ~impdar.RadarData (or its subclasses)

`load.`**load_and_exit** (*fns_in*, *channel=1*, **args*, ***kwargs*)
Load a list of files of a certain type, save them as StODeep mat files, exit

Parameters

- **filetype** (`str`) –

The type of file to load. Options are: 'pe' (pulse ekko) 'gssi' (from sir controller) 'gprMax' (synthetics) 'gecko' (St Olaf Radar) 'segy' (SEG Y) 'mcords_nc' (MCoRDS netcdf) 'mcords_mat' (MCoRDS matlab format) 'mat' (StODeep matlab format)

- **`fn`** (*`list or str`*) – List of files to load (or a single file)
- **`channel`** (*`Receiver channel that the data were recorded on`*) – This is primarily for the St. Olaf HF data

4.2.5 Processing

This defines generic processing functions to ease calls from executables.

If interacting with the API, most processing steps should probably be called by using methods on *RadarData* objects, so see *that documentation* for most of your needs. However, you may need to concatenate, which is defined separately because it acts on multiple objects.

While the `process` and `process_and_exit` directives can be used, they are generally not as useful as the direct calls.

`impdar.lib.process.`**`concat`**(*radar_data*)

Concatenate all radar data input.

Parameters **`radar_data`** (*`list of RadarData`*) – Objects to concatenate

Returns A single, concatenated output.

Return type *RadarData*

`impdar.lib.process.`**`process`**(*RadarDataList*, *interp=None*, *rev=False*, *vbp=None*, *hfilt=None*, *ahfilt=False*, *nmo=None*, *crop=None*, *hcrop=None*, *restack=None*, *denoise=None*, *migrate=None*, ***kwargs*)

Perform one or more processing steps on a list of RadarData .

Parameters

- **`RadarDataList`** (*`list of strs`*) – The *~impdar.RadarData* objects to process
- **`rev`** (*`bool, optional`*) – Reverse the profile orientation. Default is False.
- **`vbp`** (*`2-tuple, optional`*) – Vertical bandpass between (vbp1, vbp2) MHz. Default None (no filtering).
- **`hfilt`** (*`2-tuple, optional`*) – Horizontal filter subtracting average trace between (hfilt1, hfilt2). Default is None (no hfilt).
- **`ahfilt`** (*`bool, optional`*) – Adaptively horizontally filter the data.
- **`denoise`** (*`bool, optional`*) – denoising filter (only wiener for now).
- **`migrate`** (*`string, optional`*) – Migrates the data.

Returns **processed** – If True, we did something, if False we didn't

Return type bool

`impdar.lib.process.`**`process_and_exit`**(*fn*, *cat=False*, *filetype='mat'*, ***kwargs*)

Perform one or more processing steps, save, and exit.

Parameters

- **`fn`** (*`list of strs`*) – The filename(s) to process.
- **`cat`** (*`bool, optional`*) – If True, concatenate files before processing rather than running through each individually.

- **`filetype`** (*`str, optional`*) – The type of input file. Default is .mat.
- **`kwargs`** – These are the processing arguments for *process*

4.2.6 ImpdarError

`exception` `impdar.lib.ImpdarError.`**`ImpdarError`**
Used for exceptions caused by something radar-y.

4.3 Executables

ImpDAR has four executables:

impdar is a generic call that can process data, load data, or plot. Using this call, you can perform a number of processing steps in one go, saving time on loading and saving and saving disk space on not writing intermediate outputs.

impproc is designed to give greater flexibility and cleaner syntax for processing. It only performs one processing step at a time, but will thus give you intermediate outputs, by default saved with names indicating the processing performed.

impplot plots data, either as a radargram, as a line plot of power versus depth, or as the return power from a pick. It can either save the plot or bring it up for interactive panning and zooming.

imppick calls up the interpretation GUI. Some processing can also be done through this GUI.

Contents:

4.3.1 impdar

The main executable for the ImpDAR package.

```
usage: impdar [-h] {load,proc,plot,convert} ...
```

Sub-commands:

load

Load data

```
impdar load [-h] [-channel CHANNEL] [-gps_offset GPS_OFFSET] [-o O]
            [--filetype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
            {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_mat,ramac,
↪bsi,delores,osu,ramac}
            fns_in [fns_in ...]
```

Positional Arguments

filetype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file

fns_in	File(s) to load

Named Arguments

-channel	Receiver channel to load, this is primarily for the St. Olaf HF data. Default: 1
-gps_offset	Offset of GPS and data times for UoA_mat Default: 0.0
-o	Write to this filename
--filetype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file Default: “mat”

proc

Process data

```
impdar proc [-h] [-cat] [-vbp VBP VBP] [-hfilt HFILT HFILT] [-ahfilt] [-rev]
            [-nmo NMO NMO] [-crop CROP CROP CROP] [-hcrop HCROP HCROP HCROP]
            [-restack RESTACK] [-interp INTERP INTERP] [-denoise DENOISE]
            [-migrate MIGRATE] [-o O]
            fn [fn ...]
```

Positional Arguments

fn	File(s) to process

Named Arguments

-cat	Concatenate the files Default: False
-vbp	Bandpass the data vertically at low (MHz) and high (MHz)
-hfilt	Remove the average trace (average between hfilt0 and hfilt1)
-ahfilt	Adaptive horizontal filtering Default: False
-rev	Reverse profile Default: False
-nmo	Normal moveout correction. First argument is the transmitter-receiver separation. Second argument is the velocity of the radar wave (in m/s).
-crop	Crop the radar data in the travel-time direction. Args are the limit, whether to crop off [“top”, “bottom”], with limit defined in terms of [“snum”, “twtt”, “depth”]

-hcrop	Crop the radar data in the horizontal. Arguments are the limit, whether to crop off [“left”, “right], with limit defined in terms of [“tnum”, “dist”]
-restack	Restack to this (odd) number of traces
-interp	Reinterpolate GPS. First argument is the new spacing, in meters. Second argument is the filename (csv or mat) with the new GPS data
-denoise	Denoising filter (scipy wiener for now)
-migrate	Migrate with the indicated routine.
-o	Write to this filename

plot

Plot data

```
impdar plot [-h] [-s] [-yd] [-xd] [-tr TR TR] [-power POWER]
            [-spectra SPECTRA SPECTRA] [-o O] [-freq_limit FREQ_LIMIT]
            [-window WINDOW] [-scaling SCALING]
            fns [fns ...]
```

Positional Arguments

fns	File(s) to plot

Named Arguments

-s	Save file (do not plt.show()) Default: False
-yd	Plot the depth rather than travel time Default: False
-xd	Plot the dist rather than the trace num Default: False
-tr	Plot the traces in this range (line plot)
-power	Input a picked layer number to plot the RMS power for each trace in map view.
-spectra	Plot power spectral density across traces of radar profile. Input frequency bounds (MHz).
-o	Write to this filename
-freq_limit	Maximum frequeny to plot power spectral density to
-window	Type of window function to be used for the singal.periodogram() method Default: “hanning”
-scaling	Whether to plot power spectral density or power spectrum: default is spectrum Default: “spectrum”

convert

Convert filetype (lossy)

```
impdar convert [-h]
               [-in_fmt {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
               [-t_srs T_SRS]
               fns_in [fns_in ...] {shp,mat,segy}
```

Positional Arguments

fns_in	File(s) to convert
out_fmt	Possible choices: shp, mat, segy

Named Arguments

-in_fmt	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Input format type. If none, guess from extension, but be warned, we are bad at guessing!
-t_srs	Target spatial reference system (only used if out_fmt==shp). Give as EPSG number. Default: 4326

4.3.2 impproc

An executable to perform single processing steps.

This has a lot of convenience in terms of the call since you get more help with commands, more control of arguments, control over the order in which things are done, etc, but has the disadvantage of requiring a call/load/write for every step.

You can get a list of commands with `impproc -h`

For any individual command, you can get more help by running `impproc [command] -h`.

Examples

A sample workflow might be something like

```
# make directories for the output
mkdir bandpass hfilt nmo

# Vertical bandpass from 150-450MHz (loading in the raw data with the -gssi flag)
impproc vbp 150 450 -gssi *.DZT -o bandpass/

# do some horizontal filtering on that output
impproc hfilt 1000 2000 bandpass/*.mat -o hfilt
```

(continues on next page)

(continued from previous page)

```
# finally do a conversion to the time domain
impproc nmo 10 hfilt/*.mat -o nmo
```

The same processing steps can be done without separating the output into different folders. At risk of file clutter, the workflow could be

```
# Vertical bandpass from 150-450MHz (loading in the raw data with the -gssi flag)
impproc vbp 150 450 -gssi *.DZT

# do some horizontal filtering on that output
impproc hfilt 1000 2000 *_vbp.mat

# finally do a conversion to the time domain
impproc nmo 10 *_hfilt.mat

# Outputs are now sitting around with _vbp_hfilt_nmo before the extension
```

A similar example, with visualization of the outputs, is *here*.

Usage

```
usage: impproc [-h]
               {hfilt,ahfilt,rev,cat,elev,restack,rgain,agc,vbp,hbp,lp,crop,hcrop,nmo,
↪interp,geolocate,denoise,migrate}
               ...
```

Sub-commands:

hfilt

Horizontally filter the data by subtracting the average trace from a window

```
impproc hfilt [-h] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              start_trace end_trace fns [fns ...]
```

Positional Arguments

start_trace	First trace of representative subset
end_trace	Last trace of representative subset
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

ahfilt

Horizontally filter the data adaptively

```
impproc ahfilt [-h] [-o O]
               [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
               fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

rev

Reverse the data

```
impproc rev [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

cat

Concatenate the data

```
impproc cat [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

elev

Elevation correct

```
impproc elev [-h] [-o O]
             [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
             fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

restack

Restack to interval

```
impproc restack [-h] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                traces fns [fns ...]
```

Positional Arguments

traces	Number of traces to stack. Must be an odd number
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

rgain

Add a range gain

```
impproc rgain [-h] [-slope SLOPE] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-slope	Slope of linear range gain. Default 0.1 Default: 0.1

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

agc

Add an automatic gain

```
impproc agc [-h] [-window WINDOW] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-window	Number of samples to average Default: 50
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

vbp

Vertically bandpass the data

```
impproc vbp [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            low_MHz high_MHz fns [fns ...]
```

Positional Arguments

low_MHz	Lowest frequency passed (in MHz)
high_MHz	Highest frequency passed (in MHz)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

hbp

Horizontally bandpass the data

```
impproc hbp [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            low high fns [fns ...]
```

Positional Arguments

low	Lowest frequency passed (in wavelength)
high	Highest frequency passed (in wavelength)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

lp

Horizontally lowpass the data

```
impproc lp [-h] [-o O]
           [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
           low fns [fns ...]
```

Positional Arguments

low	Lowest frequency passed (in wavelength)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

crop

Crop the data in the vertical

```
impproc crop [-h] [-o O]
             [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
             {top,bottom} {snum,twtt,depth,pretrig} lim fns [fns ...]
```

Positional Arguments

top_or_bottom	Possible choices: top, bottom Remove from the top or bottom
dimension	Possible choices: snum, twtt, depth, pretrig Set the bound in terms of snum (sample number), twtt (two way travel time in microseconds), depth (m, calculated using the nmo_depth or a light speed of 1.69e8m/s if it doesn’t, or pretrig (the recorded trigger sample)
lim	The cutoff value
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

hcrop

Crop the data in the horizontal

```
impproc hcrop [-h] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              {left,right} {tnum,dist} lim fns [fns ...]
```

Positional Arguments

left_or_right	Possible choices: left, right Remove from the left or right
dimension	Possible choices: tnum, dist Set the bound in terms of tnum (trace number, 1 indexed) or dist (distance in km)
lim	The cutoff value
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

nmo

Normal move-out correction

```
impproc nmo [-h] [--uice UICE] [--uair UAIR] [--rho_profile RHO_PROFILE]
            [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            ant_sep fns [fns ...]
```

Positional Arguments

ant_sep	Antenna separation
fns	The files to process

Named Arguments

--uice	Speed of light in ice in m/s (default 1.69e8) Default: 169000000.0
--uair	Speed of light in air in m/s (default 3.0e8) Default: 300000000.0
--rho_profile	Filename for a depth density profile to correct wave velocity.
-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

interp

Reinterpolate GPS

```
impproc interp [-h] [--gps_fn GPS_FN] [--offset OFFSET] [--minmove MINMOVE]
               [--extrapolate] [-o O]
               [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
               spacing fns [fns ...]
```

Positional Arguments

spacing	New spacing of radar traces, in meters
fns	The files to process

Named Arguments

--gps_fn	CSV or mat file containing the GPS information. .csv and .txt files are assumed to be csv, .mat are mat. Default is None–use associated (presumably non-precision) GPS
--offset	Offset from GPS time to radar time Default: 0.0
--minmove	Minimum movement to not be stationary Default: 0.01
--extrapolate	Extrapolate GPS data beyond bounds Default: False
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

geolocate

GPS control

```
impproc geolocate [-h] [--extrapolate] [--guess] [-o O]
                  [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                  gps_fn fns [fns ...]
```

Positional Arguments

gps_fn	CSV or mat file containing the GPS information. .csv and .txt files are assumed to be csv, .mat are mat. Default is None–use associated (presumably non-precision) GPS
fns	The files to process

Named Arguments

--extrapolate	Extrapolate GPS data beyond bounds Default: False
--guess	Guess at offset Default: False
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

denoise

Denoising filter for the data image

```
impproc denoise [-h] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                vert_win hor_win fns [fns ...]
```

Positional Arguments

vert_win	Size of filtering window in vertical (number of samples)
hor_win	Size of filtering window in horizontal (number of traces)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

migrate

Migration

```
impproc migrate [-h] [--mtype {stolt,kirch,phsh,tk,sumigtk,sustolt,sumigffd}]
                [--vel VEL] [--vel_fn VEL_FN] [--nearfield] [--htaper HTAPER]
                [--vtaper VTAPER] [--nxpad NXPAD] [--tmig TMIG]
                [--verbose VERBOSE] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

--mtype	Possible choices: stolt, kirch, phsh, tk, sumigtk, sustolt, sumigffd Migration routines. Default: "phsh"
--vel	Speed of light in dielectric medium m/s default is for ice, 1.69e8) Default: 169000000.0
--vel_fn	Filename for inupt velocity array. Column 1: velocities, Column 2: z locations, Column 3: x locations (optional)
--nearfield	Boolean for nearfield operator in Kirchhoff migration. Default: False
--htaper	Number of samples for horizontal taper Default: 100
--vtaper	Number of samples for vertical taper Default: 1000
--nxpad	Number of traces to pad with zeros for FFT Default: 100
--tmig	Times for velocity profile Default: 0
--verbose	Print output from SeisUnix migration Default: 1

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

4.3.3 impplot

The executable syntax is described below, but look to *Plotting examples* examples for a more useful overview of what you will get out.

```
usage: impproc [-h]
               {hfilt,ahfilt,rev,cat,elev,restack,rgain,agc,vbp,hbp,lp,crop,hcrop,nmo,
↪interp,geolocate,denoise,migrate}
               ...
```

Sub-commands:

hfilt

Horizontally filter the data by subtracting the average trace from a window

```
impproc hfilt [-h] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              start_trace end_trace fns [fns ...]
```

Positional Arguments

start_trace	First trace of representative subset
end_trace	Last trace of representative subset
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

ahfilt

Horizontally filter the data adaptively

```
impproc ahfilt [-h] [-o O]
               [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
               fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o Output to this file (folder if multiple inputs)

--ftype Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac

Type of file to load (default ImpDAR mat)

Default: “mat”

rev

Reverse the data

```
impproc rev [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o Output to this file (folder if multiple inputs)

--ftype Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac

Type of file to load (default ImpDAR mat)

Default: “mat”

cat

Concatenate the data

```
impproc cat [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

elev

Elevation correct

```
impproc elev [-h] [-o O]
             [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
             fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

restack

Restack to interval

```
impproc restack [-h] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                traces fns [fns ...]
```

Positional Arguments

traces	Number of traces to stack. Must be an odd number
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

rgain

Add a range gain

```
impproc rgain [-h] [-slope SLOPE] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-slope	Slope of linear range gain. Default 0.1 Default: 0.1
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

agc

Add an automatic gain

```
impproc agc [-h] [-window WINDOW] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

-window	Number of samples to average Default: 50
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

vbp

Vertically bandpass the data

```
impproc vbp [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            low_MHz high_MHz fns [fns ...]
```

Positional Arguments

low_MHz	Lowest frequency passed (in MHz)
high_MHz	Highest frequency passed (in MHz)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat)

Default: "mat"

hbp

Horizontally bandpass the data

```
impproc hbp [-h] [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            low high fns [fns ...]
```

Positional Arguments

low	Lowest frequency passed (in wavelength)
high	Highest frequency passed (in wavelength)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

lp

Horizontally lowpass the data

```
impproc lp [-h] [-o O]
           [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
           low fns [fns ...]
```

Positional Arguments

low	Lowest frequency passed (in wavelength)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

crop

Crop the data in the vertical

```
impproc crop [-h] [-o O]
             [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
             {top,bottom} {snum,twtt,depth,pretrig} lim fns [fns ...]
```

Positional Arguments

top_or_bottom	Possible choices: top, bottom Remove from the top or bottom
dimension	Possible choices: snum, twtt, depth, pretrig Set the bound in terms of snum (sample number), twtt (two way travel time in microseconds), depth (m, calculated using the nmo_depth or a light speed of 1.69e8m/s if it doesn't, or pretrig (the recorded trigger sample)
lim	The cutoff value
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

hcrop

Crop the data in the horizontal

```
impproc hcrop [-h] [-o O]
              [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
              {left,right} {tnum,dist} lim fns [fns ...]
```

Positional Arguments

left_or_right	Possible choices: left, right Remove from the left or right
dimension	Possible choices: tnum, dist Set the bound in terms of tnum (trace number, 1 indexed) or dist (distance in km)
lim	The cutoff value
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

nmo

Normal move-out correction

```
impproc nmo [-h] [--uice UICE] [--uair UAIR] [--rho_profile RHO_PROFILE]
            [-o O]
            [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,UoA_
↪mat,ramac,bsi,delores,osu,ramac}]
            ant_sep fns [fns ...]
```

Positional Arguments

ant_sep	Antenna separation
fns	The files to process

Named Arguments

--uice	Speed of light in ice in m/s (default 1.69e8) Default: 169000000.0
--uair	Speed of light in air in m/s (default 3.0e8) Default: 300000000.0
--rho_profile	Filename for a depth density profile to correct wave velocity.
-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

interp

Reinterpolate GPS

```
impproc interp [-h] [--gps_fn GPS_FN] [--offset OFFSET] [--minmove MINMOVE]
               [--extrapolate] [-o O]
               [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
               spacing fns [fns ...]
```

Positional Arguments

spacing	New spacing of radar traces, in meters
fns	The files to process

Named Arguments

--gps_fn	CSV or mat file containing the GPS information. .csv and .txt files are assumed to be csv, .mat are mat. Default is None–use associated (presumably non-precision) GPS
--offset	Offset from GPS time to radar time Default: 0.0
--minmove	Minimum movement to not be stationary Default: 0.01
--extrapolate	Extrapolate GPS data beyond bounds Default: False
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: “mat”

geolocate

GPS control

```
impproc geolocate [-h] [--extrapolate] [--guess] [-o O]
                  [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                  gps_fn fns [fns ...]
```

Positional Arguments

gps_fn	CSV or mat file containing the GPS information. .csv and .txt files are assumed to be csv, .mat are mat. Default is None–use associated (presumably non-precision) GPS
fns	The files to process

Named Arguments

--extrapolate	Extrapolate GPS data beyond bounds Default: False
--guess	Guess at offset Default: False
-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

denoise

Denoising filter for the data image

```
impproc denoise [-h] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                vert_win hor_win fns [fns ...]
```

Positional Arguments

vert_win	Size of filtering window in vertical (number of samples)
hor_win	Size of filtering window in horizontal (number of traces)
fns	The files to process

Named Arguments

-o	Output to this file (folder if multiple inputs)

--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

migrate

Migration

```
impproc migrate [-h] [--mtype {stolt,kirch,phsh,tk,sumigtk,sustolt,sumigffd}]
                [--vel VEL] [--vel_fn VEL_FN] [--nearfield] [--htaper HTAPER]
                [--vtaper VTAPER] [--nxpad NXPAD] [--tmig TMIG]
                [--verbose VERBOSE] [-o O]
                [--ftype {mat,pe,gssi,stomat,gprMax,gecko,segy,mcords_mat,mcords_nc,
↪UoA_mat,ramac,bsi,delores,osu,ramac}]
                fns [fns ...]
```

Positional Arguments

fns	The files to process

Named Arguments

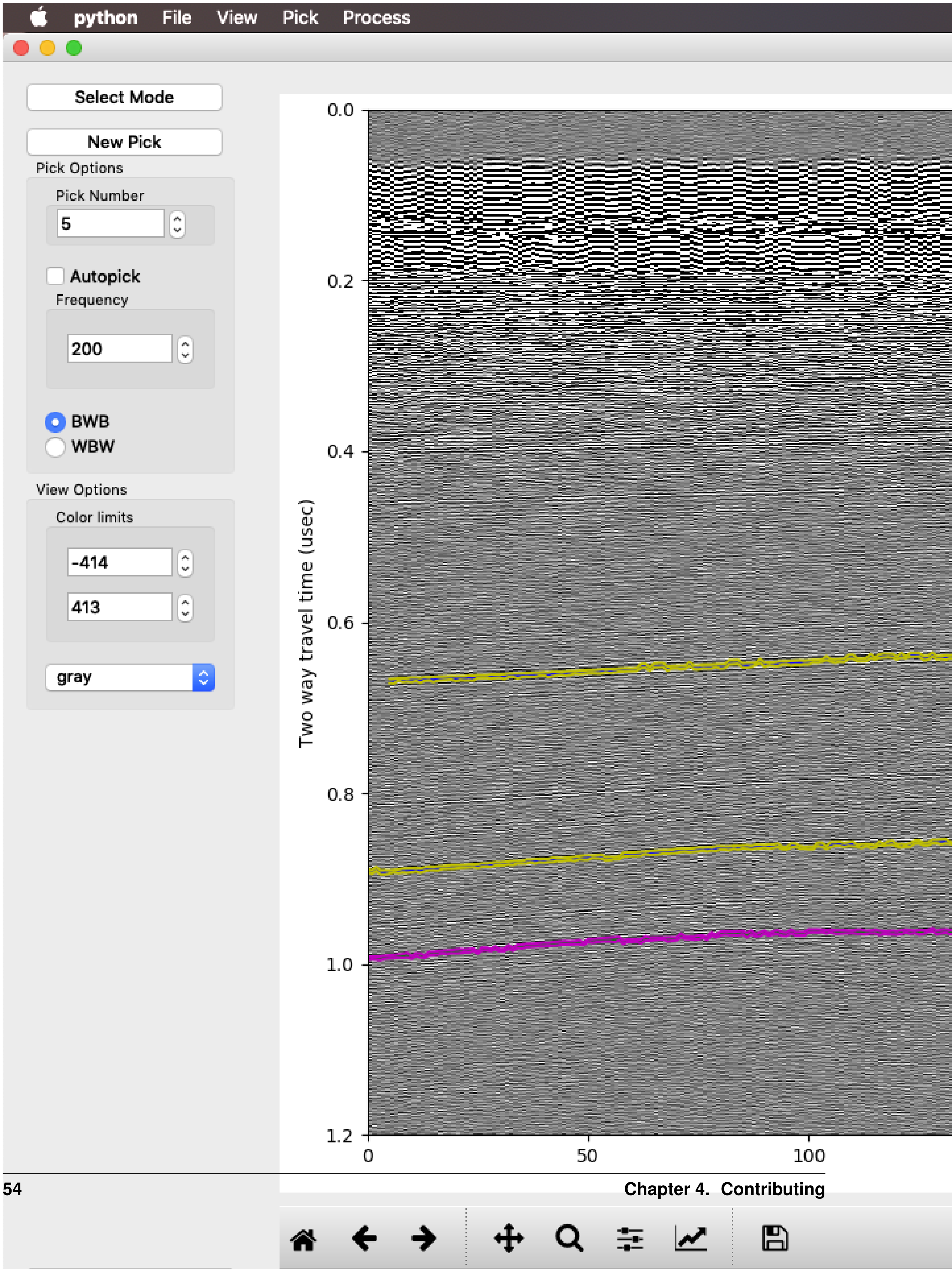
--mtype	Possible choices: stolt, kirch, phsh, tk, sumigtk, sustolt, sumigffd Migration routines. Default: "phsh"
--vel	Speed of light in dielectric medium m/s default is for ice, 1.69e8) Default: 169000000.0
--vel_fn	Filename for inupt velocity array. Column 1: velocities, Column 2: z locations, Column 3: x locations (optional)
--nearfield	Boolean for nearfield operator in Kirchhoff migration. Default: False
--htaper	Number of samples for horizontal taper Default: 100
--vtaper	Number of samples for vertical taper Default: 1000
--nxpad	Number of traces to pad with zeros for FFT Default: 100
--tmig	Times for velocity profile Default: 0
--verbose	Print output from SeisUnix migration Default: 1

-o	Output to this file (folder if multiple inputs)
--ftype	Possible choices: mat, pe, gssi, stomat, gprMax, gecko, segy, mcords_mat, mcords_nc, UoA_mat, ramac, bsi, delores, osu, ramac Type of file to load (default ImpDAR mat) Default: "mat"

4.3.4 imppick

Command-line call

After calling imppick to bring up the GUI, things should be pretty intuitive, but navigation may seem a bit odd at first. Here is the basic view of the picker on a Mac:

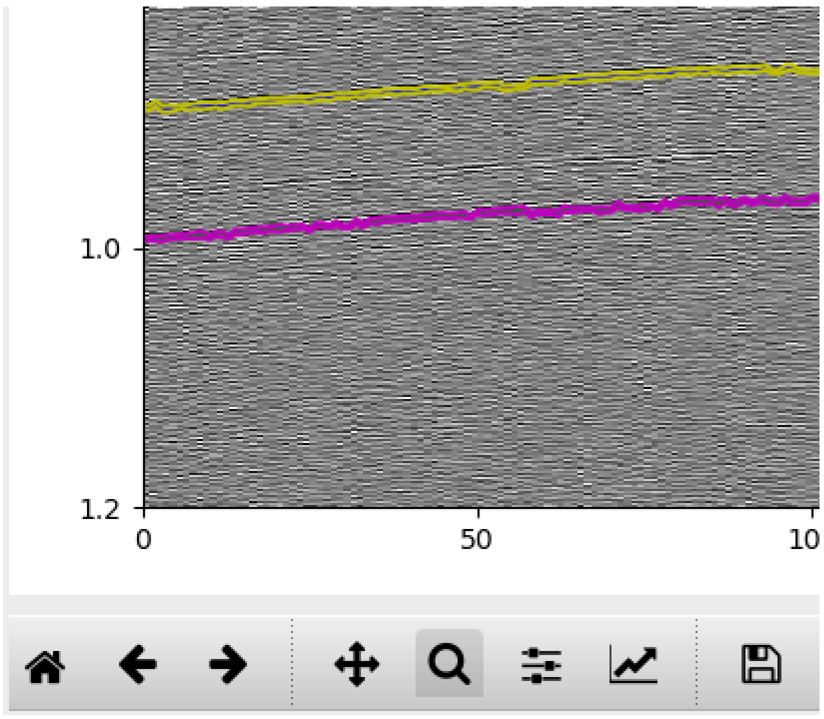

python
File
View
Pick
Process
Select Mode
New Pick
Pick Options
Pick Number
5
Autopick
Frequency
200
BWB
WBW
View Options
Color limits
-414
413
gray
Two way travel time (usec)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
50
100

In this profile, I've picked some layers already. The active pick is highlighted in magenta (or rather the top and bottom of the packet are in magenta, and the middle of the packet is green, but the middle is not visible at this zoom). Other picks are in yellow, surrounding a blue central peak. On the left side are most of the controls for when do the picking. We'll go through the buttons on the left, from top to bottom, to get an idea of how picking proceeds.

Menus

Modes

The mode is displayed in a button in the upper left. We have two modes: select mode, for deciding what to modify, and edit mode, to change the picks. **Neither mode works when the matplotlib pan/zoom toolbar is active (shown below). Reclick the zoom or pan button so it is unselected if you want to use the ImpDAR functions.**



Select

Select mode allows us to choose which of the picks to add to. This is used to go back to old picks that already exist and modify them. If there are no picks yet, or if we want a new pick, we can go straight to edit mode.

Edit

Edit mode is where you will spend most of your time. In edit mode, you can modify the existing picks, either deleting from the, renumbering them, or adding to them.

Pick Options

This is where we control things about how the picking algorithm operates.

Pick Number

Changing this integer will change the number associated with this pick. This changes nothing about how the data are stored (i.e. you can choose pick 999 without making a big empty matrix waiting for picks 1-998), and only affects wheat we call it (and it will be exported with the number of your choice). By convention, StODeep used 99 for the bed pick. Trying to set the number to something that is already used is not allowed–ImpDAR will increment to an unused pick. If you want to switch the numbering of two picks, you should set one to something unused, move the second to the first, then the first into the second's old place.

Autopick

Right now, this checkbox is inactive. If we can successfully implement a decent autopicking algorithm, this will get turned on. For now, if you want to try to make ImpDAR do the work for you, try picking the leftmost and the rightmost trace.

Frequency

This should, in general, be the frequency of the radar system. It sets the wavelet size that we try to correlate with each radar trace when picking. You probably want to update this once at the start of picking, then leave it alone.

Polarity

Choose whether you are picking layers that go +-+ or -+-. In a grayscale colorbar, these BWB and WBW respectively.

New Pick

After we have selected our picking options, we probably want to do some picking. Clicking the "New pick" button adds a pick with an unused pick number (you can modify it at any time though).

View Options

These options control aspects of coloring the radargram; zooming and panning are handled directly by matplotlib in the bottom toolbar.

Color limits

The color limits are fairly self explanatory. Change this to increase or decrease contrast.

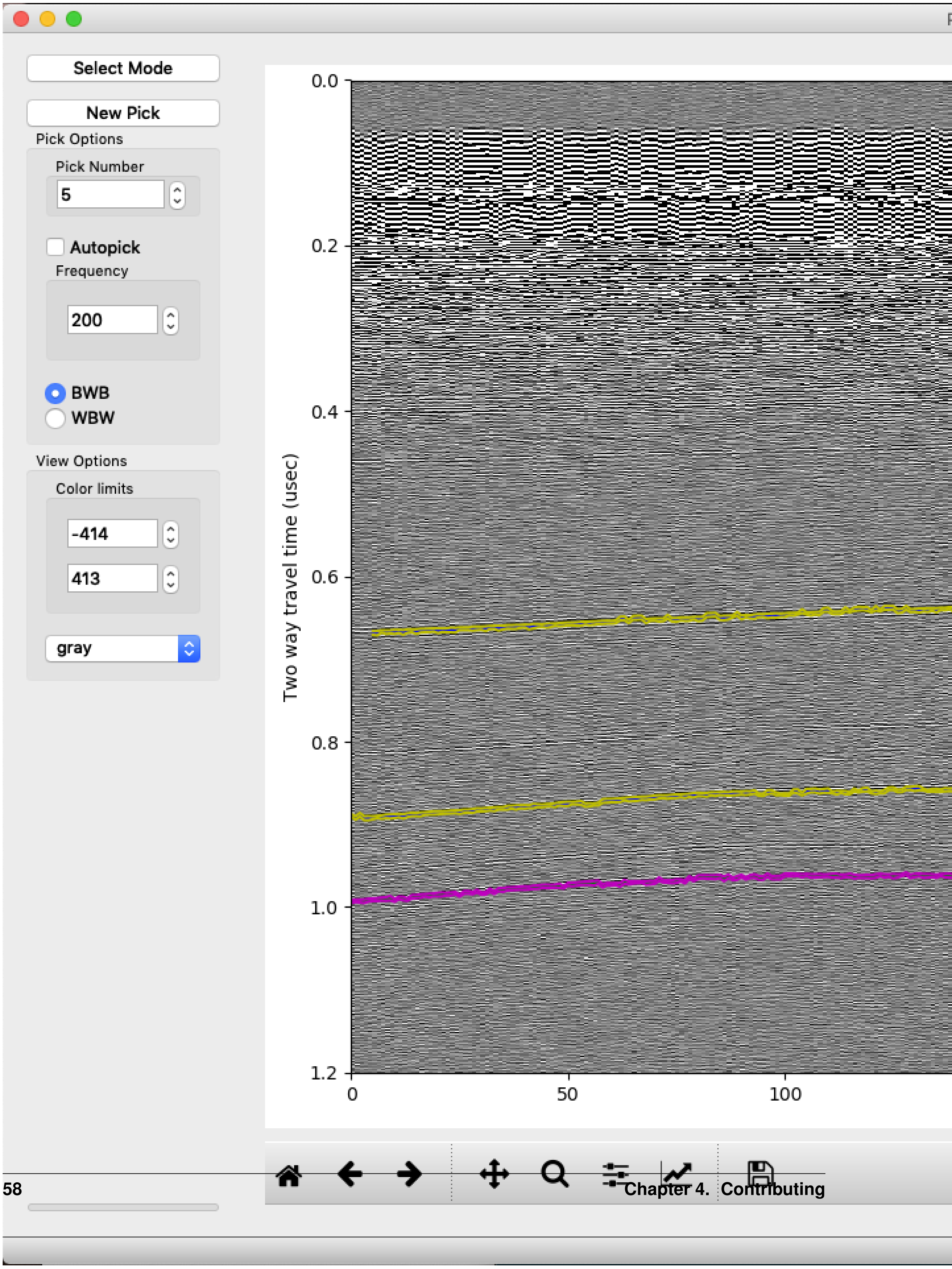
Color map

This is again self explanatory. Change the colormap as desired to improve the visualization. CEGSIC is a custom map intended for radar data and developed at St. Olaf. All other maps just link to matplotlib.

Workflow

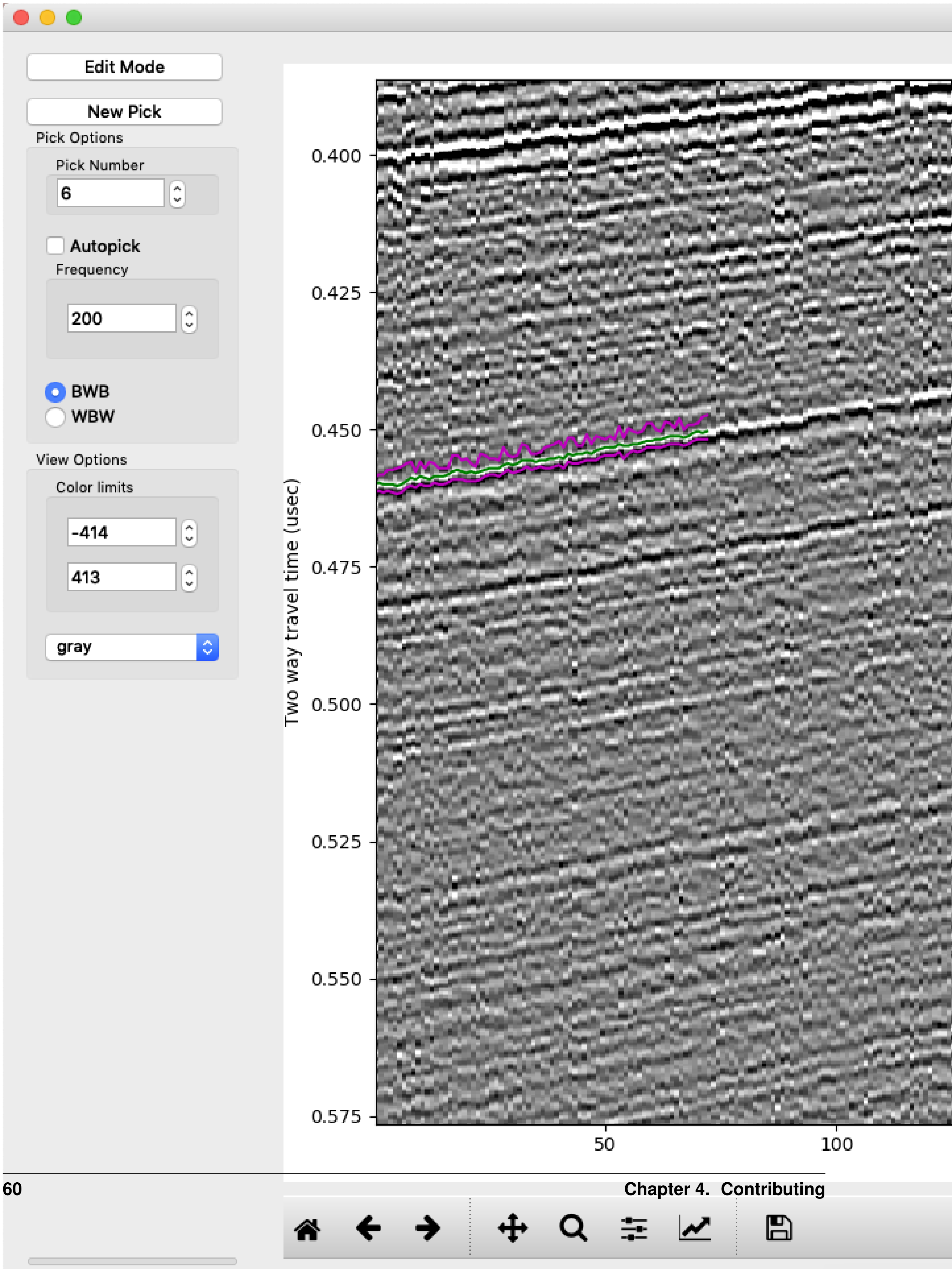
Load intersections

Once the profile is loaded, before doing any picking or numbering, you likely want to have the context of any other profiles that you have already picked. This is done through *pick > load crossprofile*. Loading the intersections should give you a string of pearls with pick numbers in each, with the dots located at where the other profile hits this one. The loading is pretty dumb, so if you have multiple intersections only the one where the traces in the two profiles are closest will load. Eventually this might become more clever, but the current implementation covers most use cases. You can load multiple profiles, so if you are really having a need for multiple intersections, just split the other file.

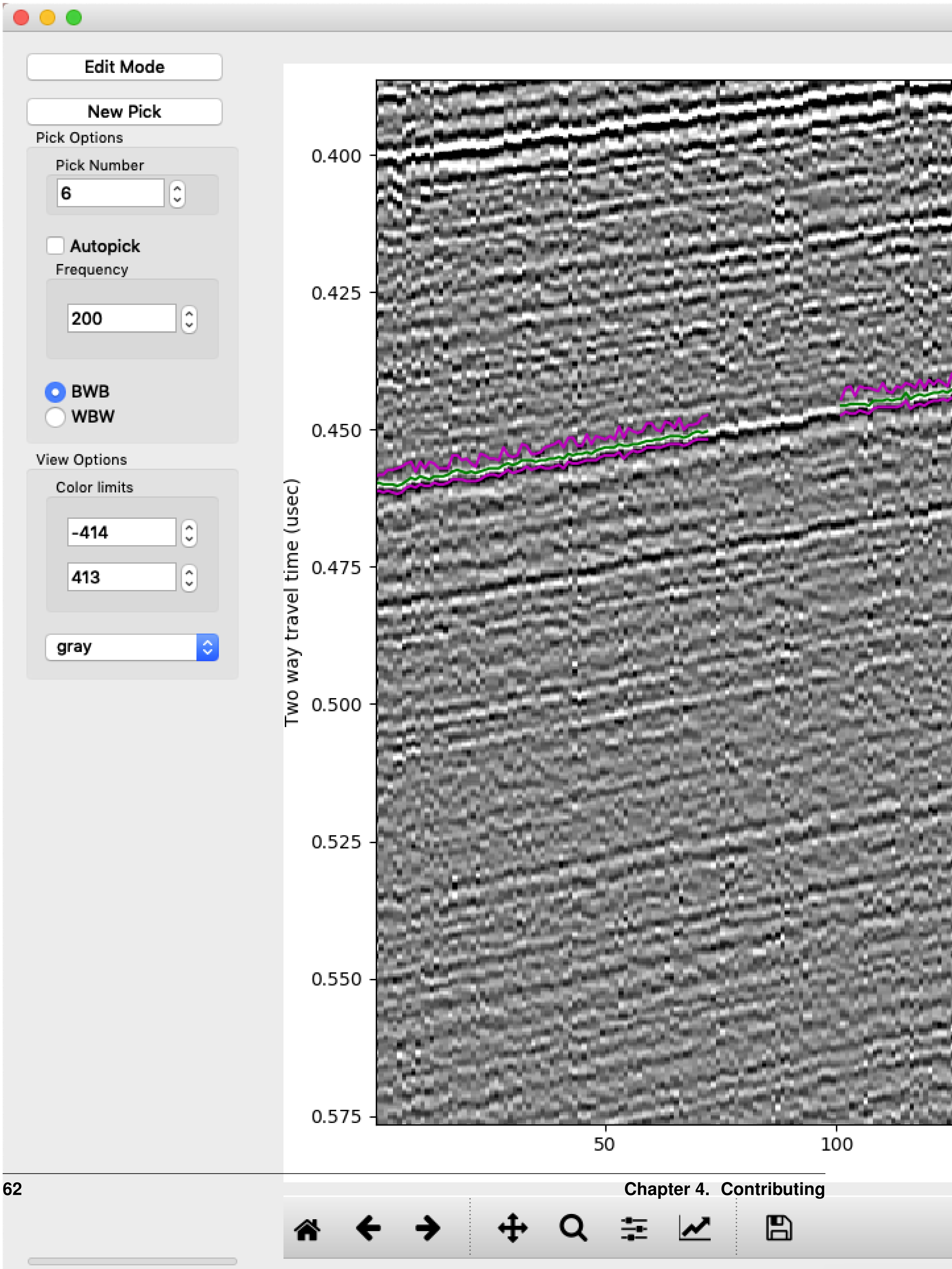

Select Mode
New Pick
Pick Options
Pick Number
5
Autopick
Frequency
200
BWB
WBW
View Options
Color limits
-414
413
gray
Two way travel time (usec)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
50
100

Picking

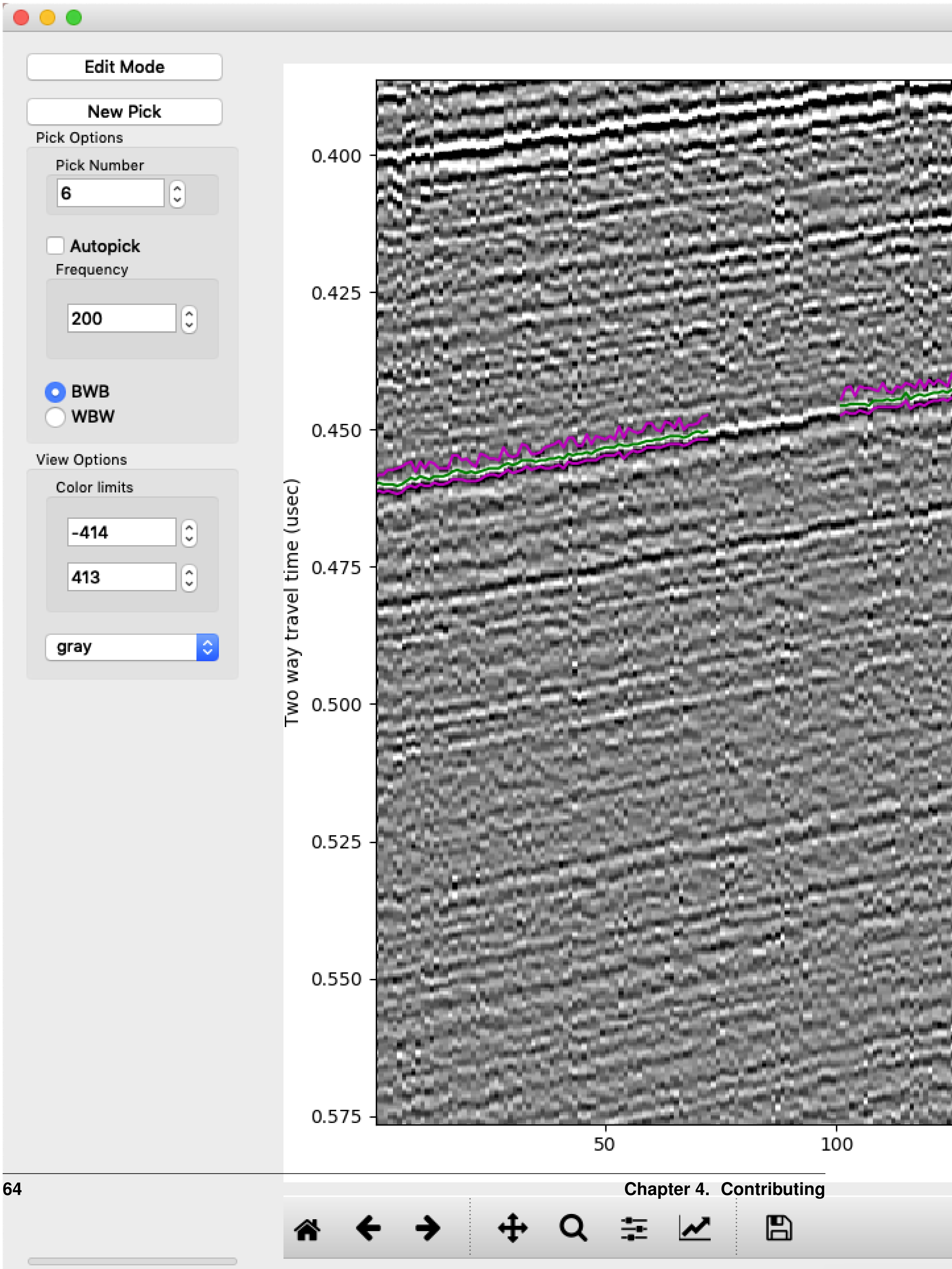
To begin picking, make sure you are in "edit" mode and that neither pan nor zoom is selected. If there are already some picks on the profile, you first will want to create a new pick. Picking a section must be done from left to right. You can skip portions by "NaN picking", then continue to the right and go back and fill in gaps later to fill in gaps. To pick, start with a left click on the layer at the left edge of the profile. After you click a second time, you should start to see the layer plotted. You should not try to pick too far away–ImpDAR will search for a reflection with the desired polarity within a certain distance, determined by the frequency, of the line connecting your clicks. If you try to make it pick through too much variability, it can miss peaks and troughs.


Edit Mode
New Pick
Pick Options
Pick Number
6
Autopick
Frequency
200
BWB
WBW
View Options
Color limits
-414
413
gray
Two way travel time (usec)
0.400
0.425
0.450
0.475
0.500
0.525
0.550
0.575
50
100

Now, let's say you come to a portion of the profile that you feel is ambiguous and you want to skip it. Pick up to the left side of it, then click on the right side while holding down "n". Continue clicking to the right as normal, and you will see that the portion left of where you clicked with "n", i.e. where you NaN picked, is blank.


Edit Mode
New Pick
Pick Options
Pick Number
6
Autopick
Frequency
200
BWB
WBW
View Options
Color limits
-414
413
gray
0.400
0.425
0.450
0.475
0.500
0.525
0.550
0.575
Two way travel time (usec)
50
100

Now suppose you screwed up, like in the image above where it looks like you stepped down to a deeper layer by mistake, so now you want to backtrack. Right clicking will delete all picks left of the last click (generally the right end of the profile) and right of the right click.


Edit Mode
New Pick
Pick Options
Pick Number
6
Autopick
Frequency
200
BWB
WBW
View Options
Color limits
-414
413
gray
Two way travel time (usec)
0.400
0.425
0.450
0.475
0.500
0.525
0.550
0.575
50
100

We can also go back and edit a previous pick, moving it up, say. We can also delete picks in the middle of a profile by left clicking at the right edge of the span we are deleting, then right click at the left edge.

Saving

After picking, you need to save your picks. When you close the window, you will be prompted to save. You can also save at any time through the file menu in the upper left. If you just want to save an image of the figure, you can use the disk icon in the matplotlib toolbar or you can use the *file* > *save figure* from the menus. You can also export the picks as a csv file (no gdal required) or as a shapefile (needs gdal) from the *pick* > *export* menu.

4.4 Examples

4.4.1 Jupyter notebooks

We have set up several Jupyter notebooks so that you can interactively run through code.

The code and output can be seen through this website. To truly run the code and modify it, you can get the source repository here. Some of the data source files are a bit large (~100 Mb) to give realistic examples, so this repository is separated from the main code.

ImpDAR Getting Started Tutorial

Welcome to ImpDAR! This is an impulse radar processor primarily targeted toward ice-penetrating radar systems. However, the potential applications of this software are much more widespread. Our goal is to provide an alternative to the expensive commercial software that would be purchased with a ground-penetrating radar system from a company such as GSSI or Sensors and Software. We provide all of the processing tools that are needed to move from a raw data file to an interpretable image of the subsurface. Moreover, our software is agnostic to the system used, so we can import data from a variety of different ground-penetrating radar systems and interact with the data in the exact same way.

If you have not yet installed ImpDAR on your computer, you can go to our github page (https://github.com/dlilien/ImpDAR) to download or clone the repository. For those less familiar with Python programming, take a look at our Read the Docs page for help (https://impdar.readthedocs.io/en/latest/).

The preferred pathway for interacting with ImpDAR is through the terminal. However, our API allows relatively easy access through other programs as well. Here, we want to walk you through the processing flow using a Jupyter Notebook, where all of the processing functions can be called through Python. No prior knowledge of Python is necessary.

Import the necessary libraries

The very first thing to do with any Python script is to import all of the libraries that will be used. These will allow us to call functions that help with loading scripts, some numerical issues, and plotting. Knowing exactly what each of these are is not important to understanding the radar processing flow.

[1]:
```
# We get annoying warnings about backends that are safe to ignore
import warnings
warnings.filterwarnings('ignore')

# Standard Python Libraries
import numpy as np
```

(continues on next page)

(continued from previous page)

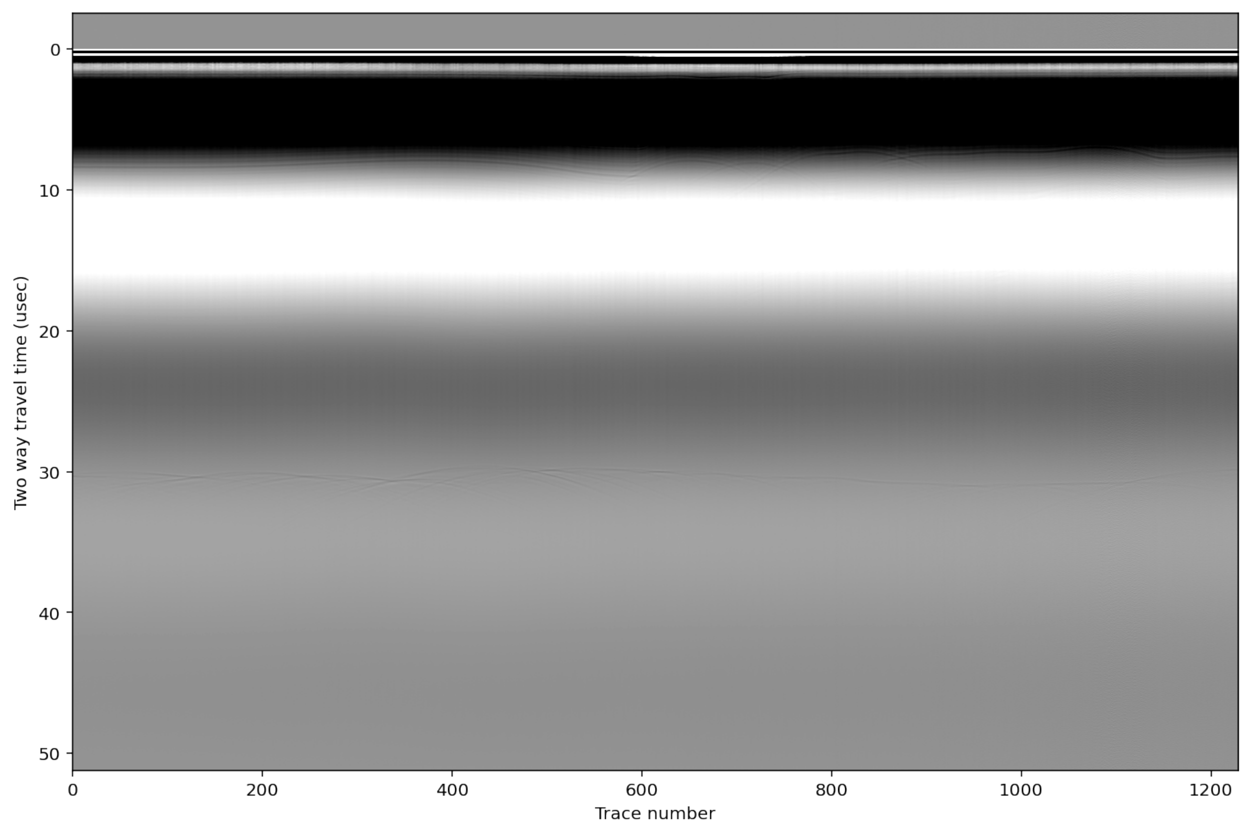
```
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
%config InlineBackend.figure_format = 'retina'
```

Load the raw data

With the standard libraries loaded, we can now look at some radar data. This particular radargram was collected at the Northeast Greenland Ice Stream (Christianson et al., 2014). We discuss more of the details about exactly what we are looking at when we get to a more interpretable image.

[2]:
```
# To look through data files we use glob which is a library
# that finds all the file names matching our description
import glob
files = glob.glob('data_in/*[!.gps]')

# Impdar's loading function
from impdar.lib import load
dat = load.load_olaf.load_olaf(files)
# save the loaded data as a .mat file
dat.save('test_data_raw.mat')

# Impdar's plotting function
from impdar.lib.plot import plot_traces, plot_radargram
%matplotlib inline
plot_radargram(dat)
```

[2]:
```
(<Figure size 864x576 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910f0cd6a0>)
```

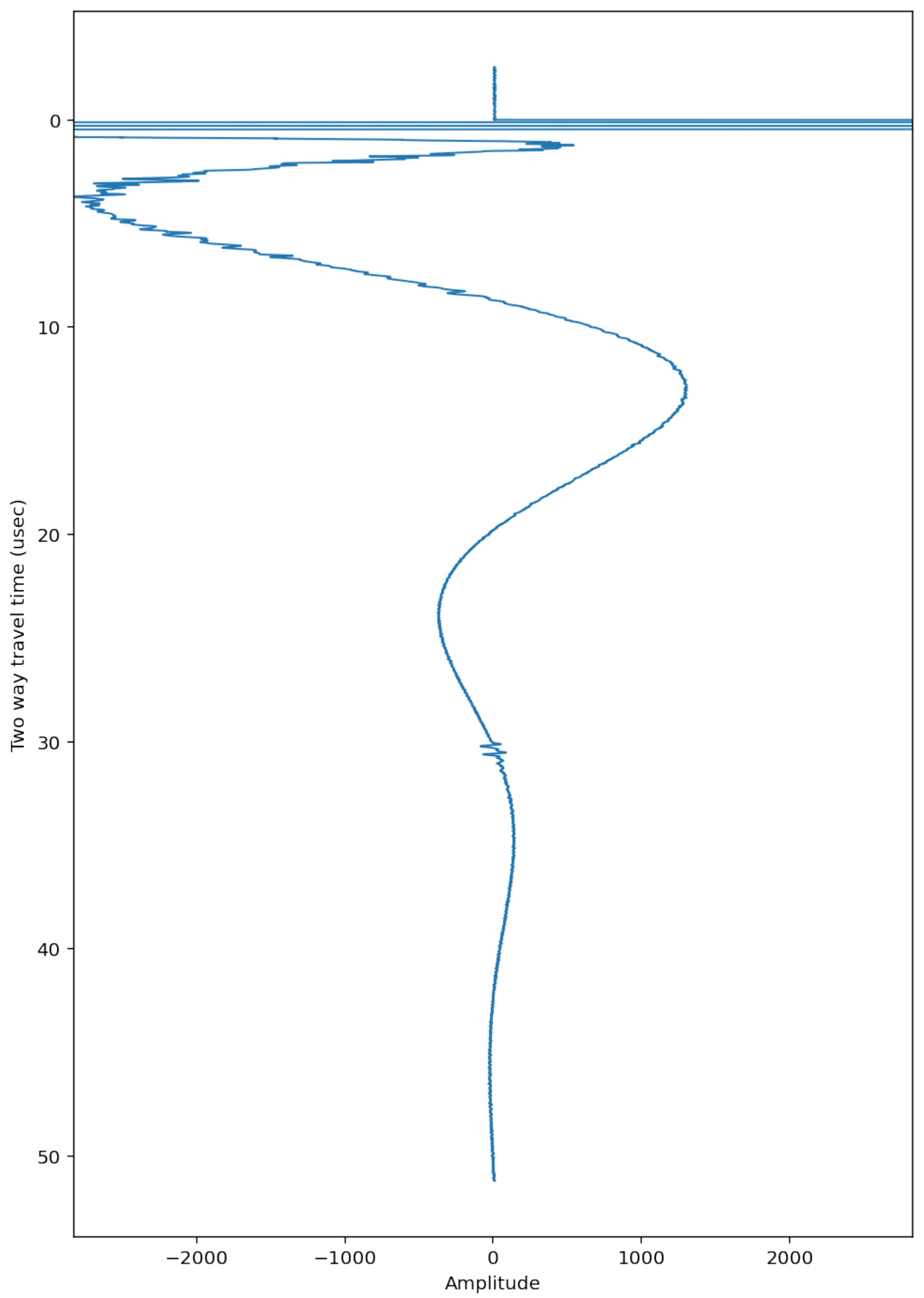


Raw radar data are a mess. We can't really see anything meaningful here.

In order to illustrate the first processing step, let's plot a single 'trace'. A radar trace is one collection of voltages measured by the oscilloscope through time. The total time for collection in this case is ~50 microseconds.

[3]:
```
### Try changing the trace index to see whether they are similar or different ###
### Make sure that you understand the relationship between this figure and above ###

# Reload in case a lower cell gets run
dat = load.load_olaf.load_olaf(files)

# Plot a single trace, where tr is the trace number in the above image
plot_traces(dat,tr=100)
```

[3]:
```
(<Figure size 576x864 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910c8ff978>)
```



Bandpass filter

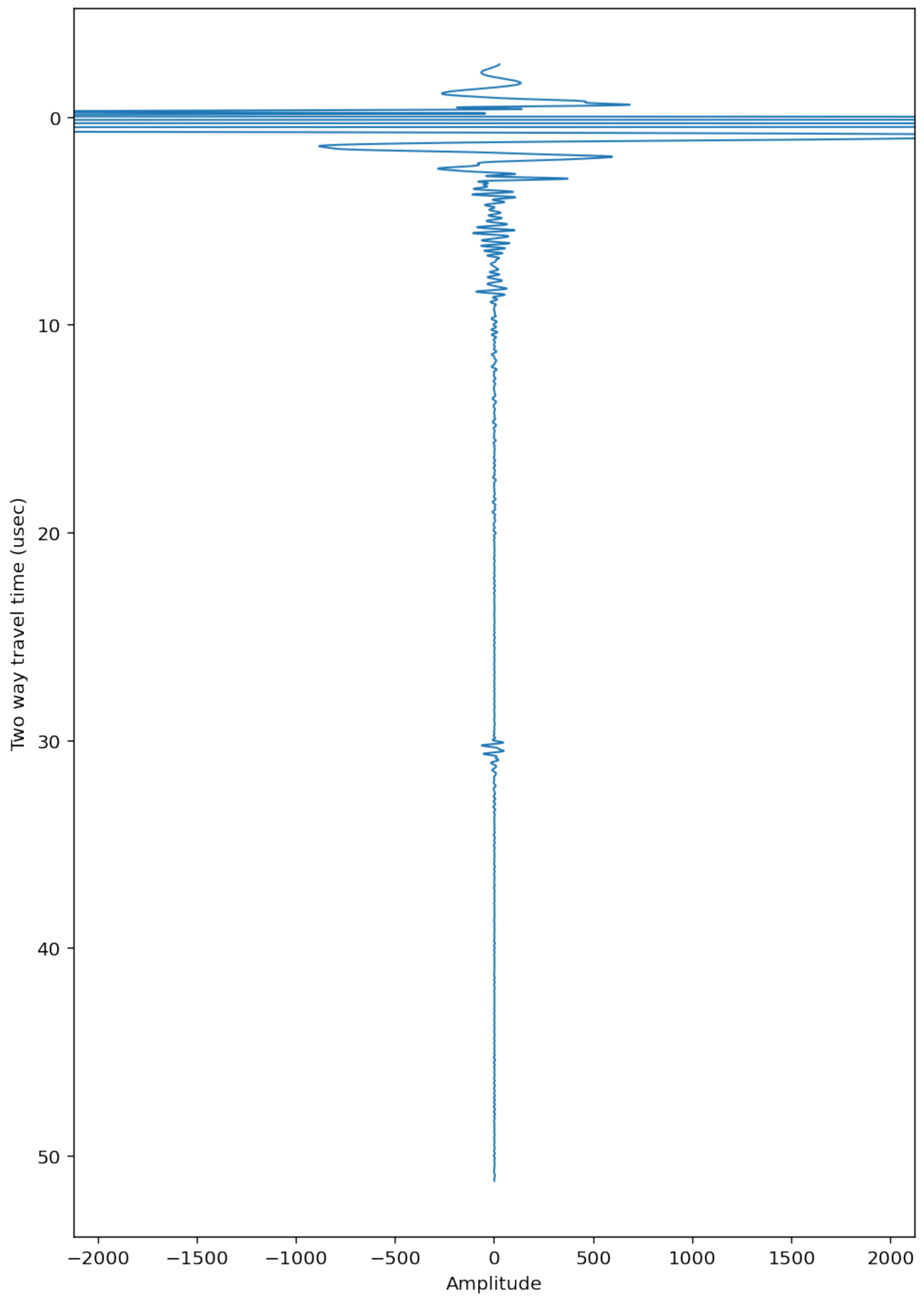
The first big processing step is to isolate the frequency band of interest. The radar antennas set this frequency. In our case, the antennas are ~20 m long (frequency of 3 MHz) so we want to allow the frequency band from 1-5 MHz pass while damping all other frequencies (i.e. bandpass filter).

This filter works on all the traces simultaneously, but to illustrate its effect we will plot the single trace again.

[4]:
```
# Do the vertical bandpass filter from 1-5 MHz
dat.vertical_band_pass(1,5)
# save again
dat.save('./test_data_bandpassed.mat')

# Plot a single trace
plot_traces(dat, tr=100)
```

```
Bandpassing from  1.0 to  5.0 MHz...
Bandpass filter complete.
```

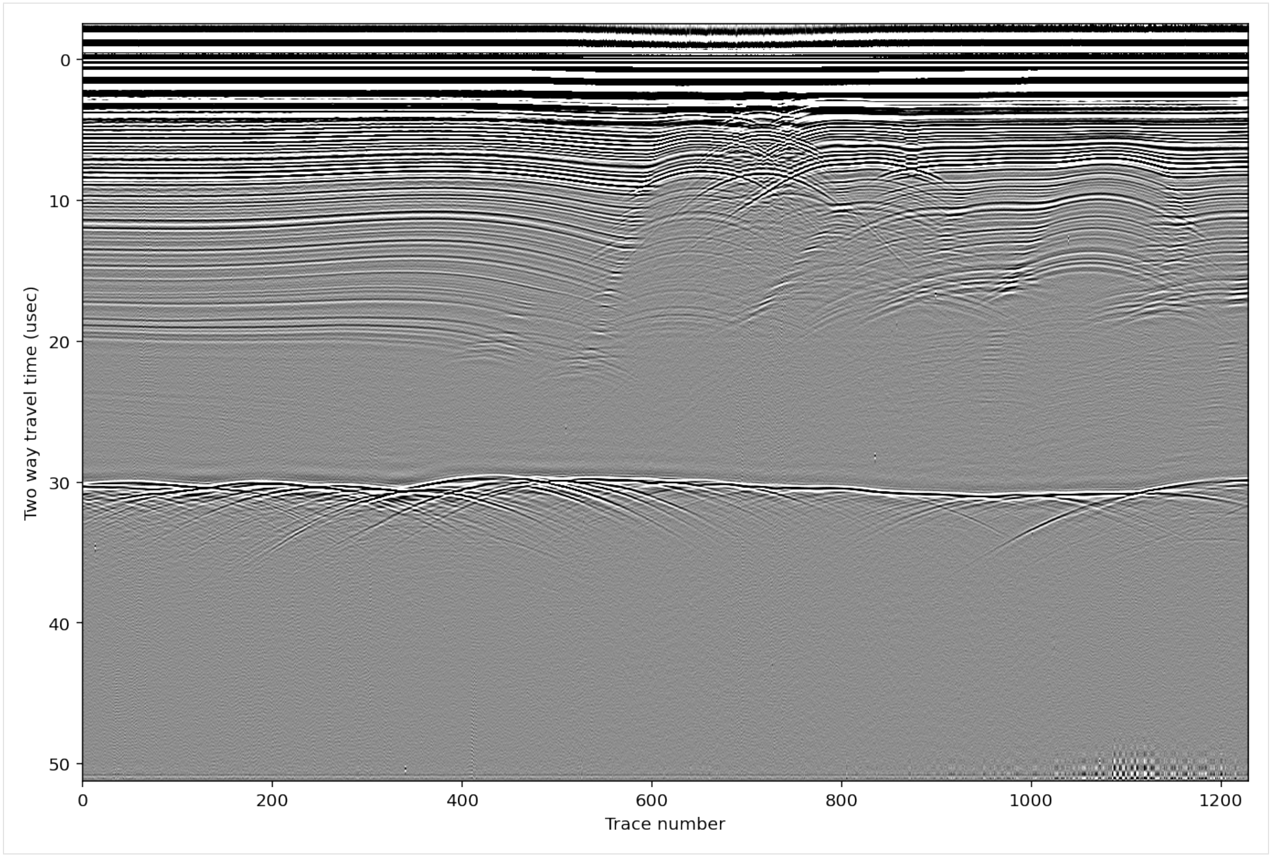
[4]:
```
(<Figure size 576x864 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910c8ff668>)
```



Now plot the entire image again.

```
[5]: # Plot all the bandpassed data
plot_radargram(dat)
```

```
[5]: (<Figure size 864x576 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910c7fd1d0>)
```



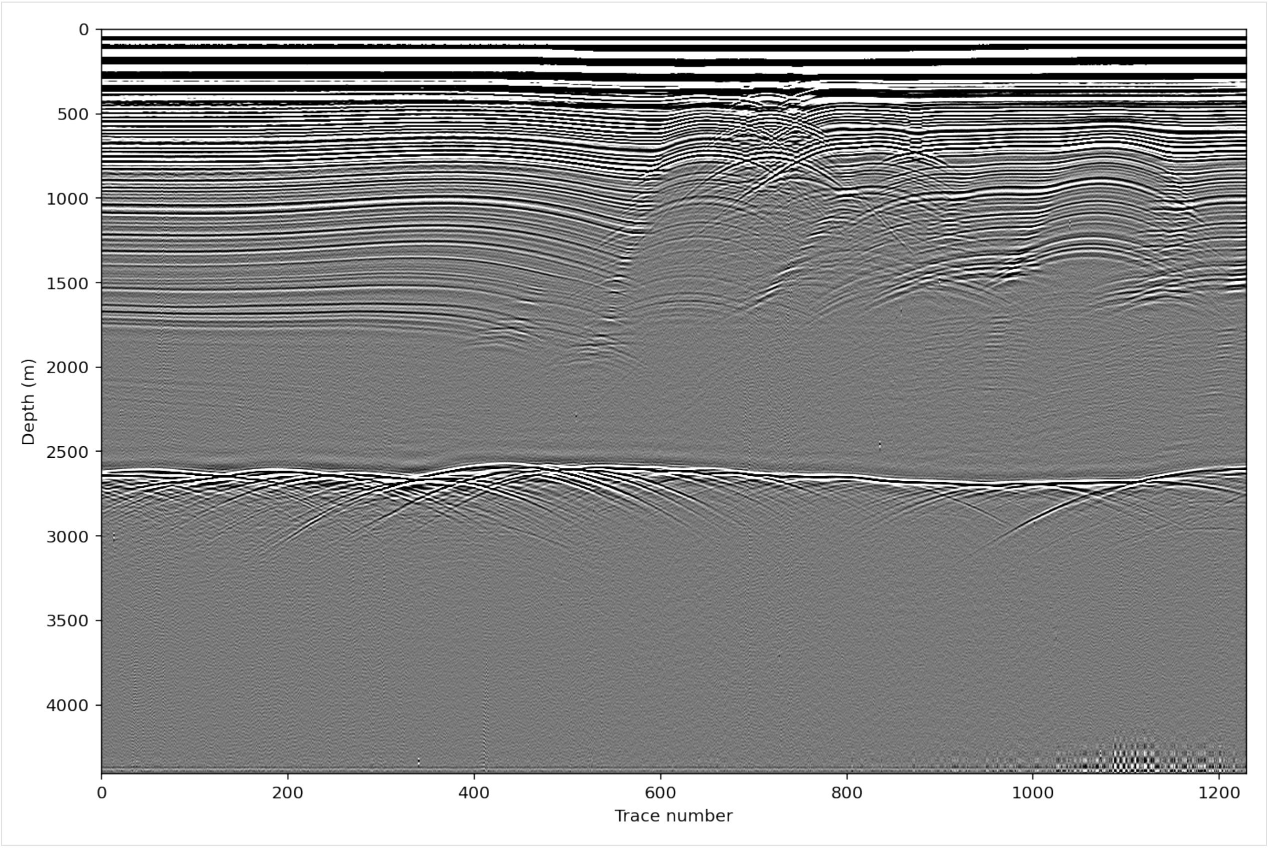
Geometric Corrections

Now that the data look like something useful, we can do a few more corrections to make it more physical. Two geometric corrections allow us to look at the image more like something under the surface instead of in this two-way travel time dimension that we have been using. First we crop out the 'pretrigger' which is data collected by the receiver before the radar pulse is actually transmitted. Second we do a 'normal move-out' correction (nmo) which corrects for the separation distance between the receiving and transmitting antennas. We have an additional tutorial for more details about the nmo filter in the case of a variable velocity (e.g. in snow or firn). After the nmo correction, we can more responsibly plot the y axis in 'depth' rather than 'travel time' by adding 'yd=True' in the plot function.

```
[6]: # Crop the pretrigger out from the top of each trace
dat.crop(0.,dimension='pretrig')
# Do the normal move-out correction with antenna spacing 160 m
dat.nmo(160)

# Save and plot
dat.save('test_data_nmo.mat')
plot_radargram(dat, ydat='depth')
```

```
Vertical samples reduced to subset [512:10752] of original
```

```
[6]: (<Figure size 864x576 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910c7daa90>)
```



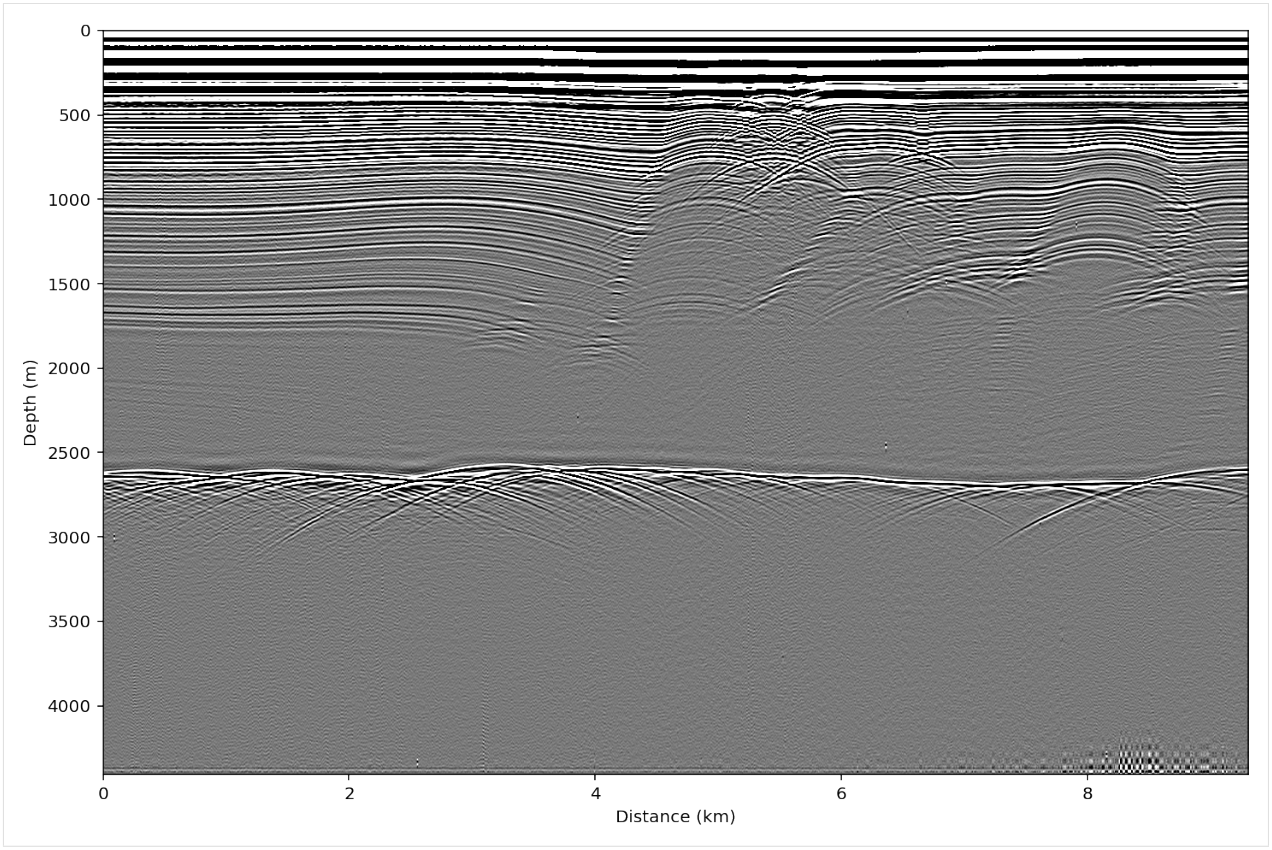
Georeferencing and Interpolation

Up to now, everything has been plotted as ‘trace number’ in the horizontal. In reality though, we want to know where in space each trace was measured. Typically, these data are collected with some kind of GPS either internal to the system itself, or attached as an external antenna.

In cases where the GPS data are available, ImpDAR can load the data in, assigning lat/long to each trace. Then, the coordinates can be projected into x/y and a distance vector created for distance traversed along the profile. The final step is to interpolate the image onto equal trace spacing. All this is handled in the ‘interp’ function as seen below.

[7]:
```
# Interpolate onto equal trace spacing of 5 m
from impdar.lib.gpslib import interp
interp([dat],5)

# Save and plot the interpolated image
dat.save('test_data_interp.mat')
plot_radargram(dat,ydat='depth',xdat='dist')
```

[7]:
```
(<Figure size 864x576 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f910c790be0>)
```



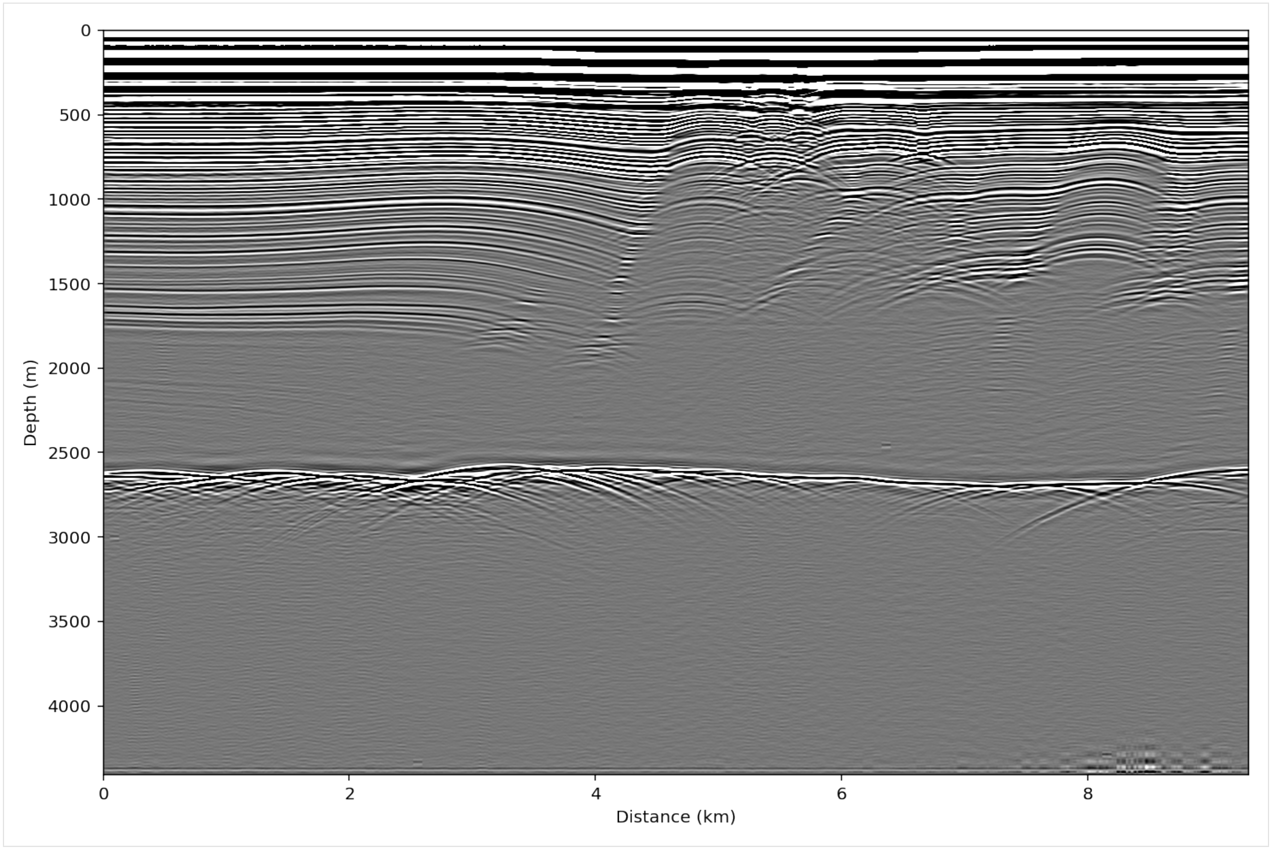
Denoise

Denoising is done with a 2-dimensional Wiener filter. Inputs are the number of pixels to include in the filter (vertical, horizontal).

**Note: After denoising the image, the wave amplitude no longer has a physical meaning. For amplitude analysis, denoising filters should be avoided.

```
[8]: dat.denoise(1,15)

     # Save and plot
     dat.save('test_data_denoise.mat')
     plot_radargram(dat,ydat='depth',xdat='dist')
```

```
[8]: (<Figure size 864x576 with 1 Axes>,
      <matplotlib.axes._subplots.AxesSubplot at 0x7f910c720748>)
```



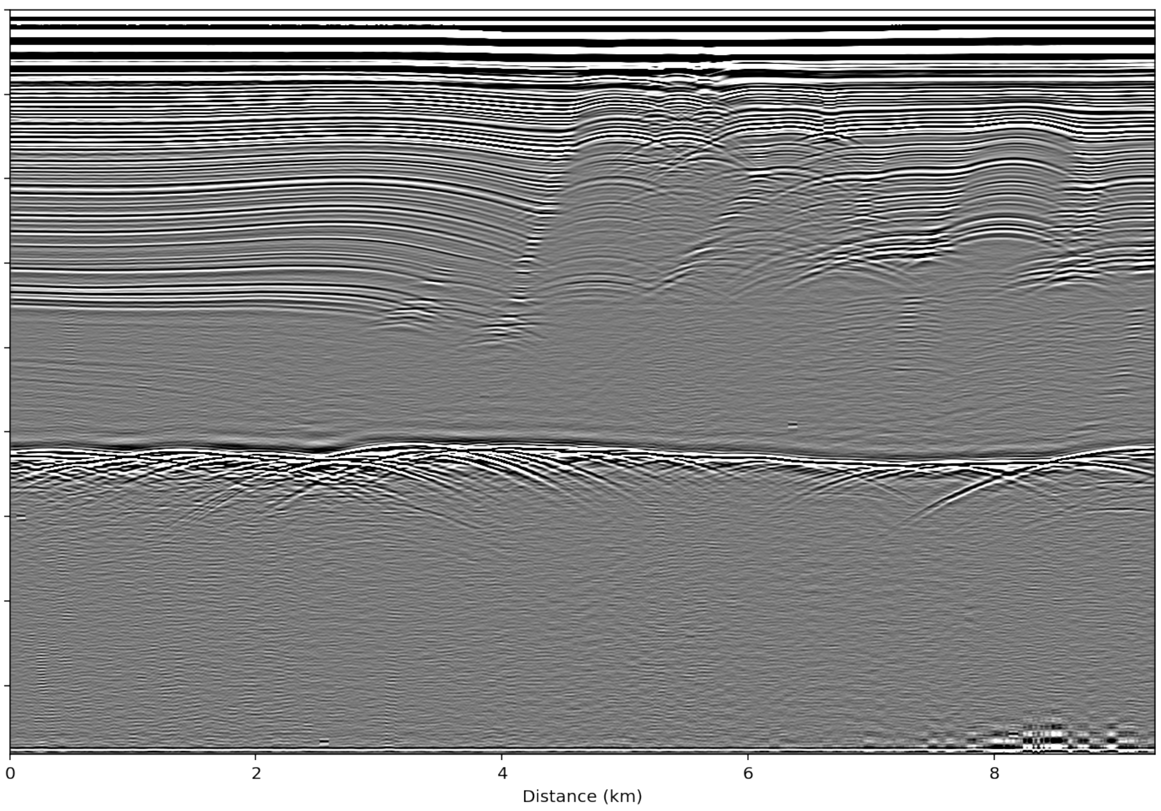
Linear Range Gain

Sometimes, when the data are very low amplitude near the bed, we want to boost the signal so that we can see it. The simplest way to do this is with a linear range gain (i.e. multiply each trace by a ramp that increases toward the bottom).

**Note: As with the denoising filter above, amplitude interpretations should not be done after this type of filter has been used.

```
[9]: dat.rangegain(0.05)

     # Save and plot
     dat.save('test_data_gain.mat')
     plot_radargram(dat,ydat='depth', xdat='dist')
```

```
[9]: (<Figure size 864x576 with 1 Axes>,
      <matplotlib.axes._subplots.AxesSubplot at 0x7f910c6e0d30>)
```



ImpDAR NMO Filter Tutorial (Variable Velocity)

The normal move-out filter corrects for source-receiver antenna separation by finding the legs of the triangular travel path. Within ImpDAR, this filter also converts from two-way-travel-time to depth. Both the filter and the depth conversion account for variable wave speed when requested.

Here, we walk through an example using ground-based snow radar from South Cascade Glacier. Density cores drilled in the snowpack are used to constrain the wave speed.

[1]:
```
# We get annoying warnings about backends that are safe to ignore
import warnings
warnings.filterwarnings('ignore')

# Load standard libraries
import numpy as np
import matplotlib.pyplot as plt
from impdar.lib import load
# To make the plots look nicer
plt.rcParams['figure.dpi'] = 300
%config InlineBackend.figure_format = 'retina'
```

Load the raw data (PulseEKKO format) and do some initial processecing steps (vertical bandpass and pretrigger crop) before considering the nmo filter.

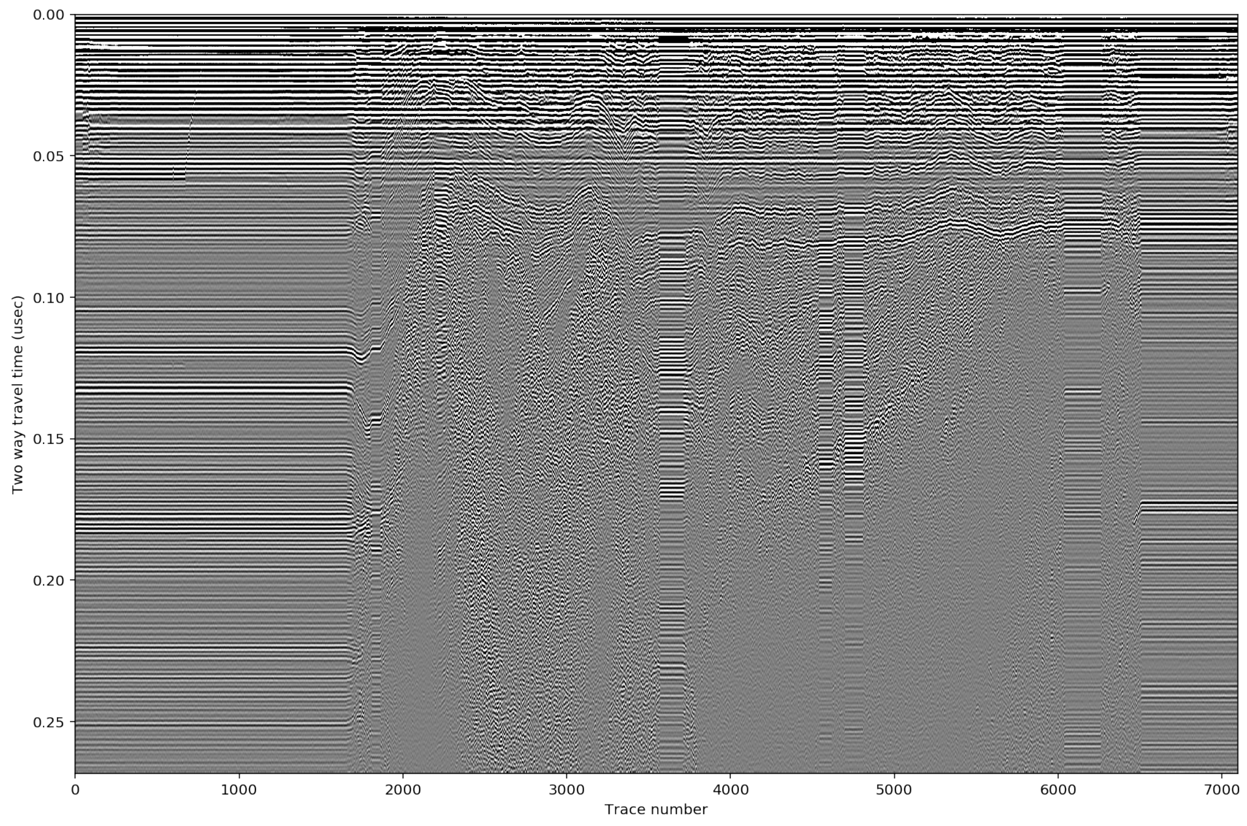
[2]:
```
# Load the Pulse EKKO data
dat = load.load('pe',['data/LINE01.DT1'])[0]
# Bandpass filter
```

(continues on next page)

(continued from previous page)

```
dat.vertical_band_pass(250,750)
# Crop the pretrigger
dat.crop(0.,dimension='pretrig',rezero=True)

# Plot the
from impdar.lib.plot import plot
%matplotlib inline
dat.save('./scg_data_raw.mat')
plot('./scg_data_raw.mat')
```

```
Bandpassing from 250.0 to 750.0 MHz...
Bandpass filter complete.
Vertical samples reduced to subset [317:3000] of original
```



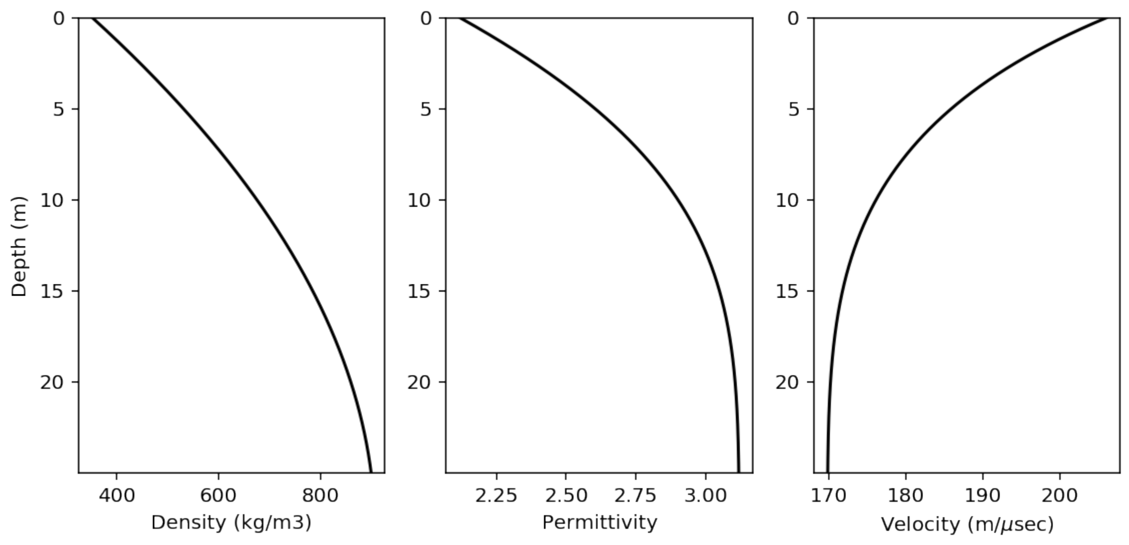
Get permittivity and wave-speed profile

The speed of light in ice is density-dependent, it travels faster in lower density snow/firn/ice which makes sense because that is closer to air. ImpDAR has some functionality to convert from snow/firn density to permittivity (wave speed). Here, we load the density profile measured at South Cascade Glacier and convert that to permittivity and subsequently to wave speed.

At every reflector depth for the correction, the nmo filter uses a root-mean-square wave speed to calculate the time between the two antennas. This is one leg of the triangle, the recorded time is the hypoteneuse, and the second leg is the vertical time that we really want.

[3]:
```
# load the density-to-permittivity function from ImpDAR
from impdar.lib.permittivity_models import firn_permittivity

# Load data from the density core at South Cascade Glacier
rho_profile_data = np.genfromtxt('2018_density_profile.csv',delimiter=',')
profile_depth = rho_profile_data[:,0]
profile_rho = rho_profile_data[:,1]
# speed of light in vacuum
c = 300.
# convert density to profile velocity
profile_vel = c/np.sqrt(np.real(firn_permittivity(profile_rho)))

# Plot a figure to show density, permittivity, and velocity profiles
plt.figure(figsize=(8,4))

ax1 = plt.subplot(131)
plt.ylabel('Depth (m)')
plt.xlabel('Density (kg/m3)')
plt.plot(profile_rho,profile_depth,'k')
plt.ylim(max(profile_depth),0)

ax2 = plt.subplot(132)
plt.xlabel('Permittivity')
plt.plot(firn_permittivity(profile_rho),profile_depth,'k')
plt.ylim(max(profile_depth),0)

ax3 = plt.subplot(133)
plt.xlabel('Velocity (m/$\mu$sec)')
plt.plot(profile_vel,profile_depth,'k')
plt.ylim(max(profile_depth),0)

plt.tight_layout()
plt.draw()
```

```
/Users/dlilien/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:␣
↪ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



NMO Correction

Finally, do the nmo correction. This correction stretches samples near the surface, so a 1-dimensional interpolation is done afterward to get equal depth sampling.

You will notice that the corrected times are longer than the uncorrected times. This is because the raw times are measured at the reciever. Really though, we need the times relative to the transmit pulse for the correction. The time for the initial pulse to get to the antennae is added to the measured time to get the transmitted time and that is corrected to get the vertical leg of the triangle (the final nmo time).

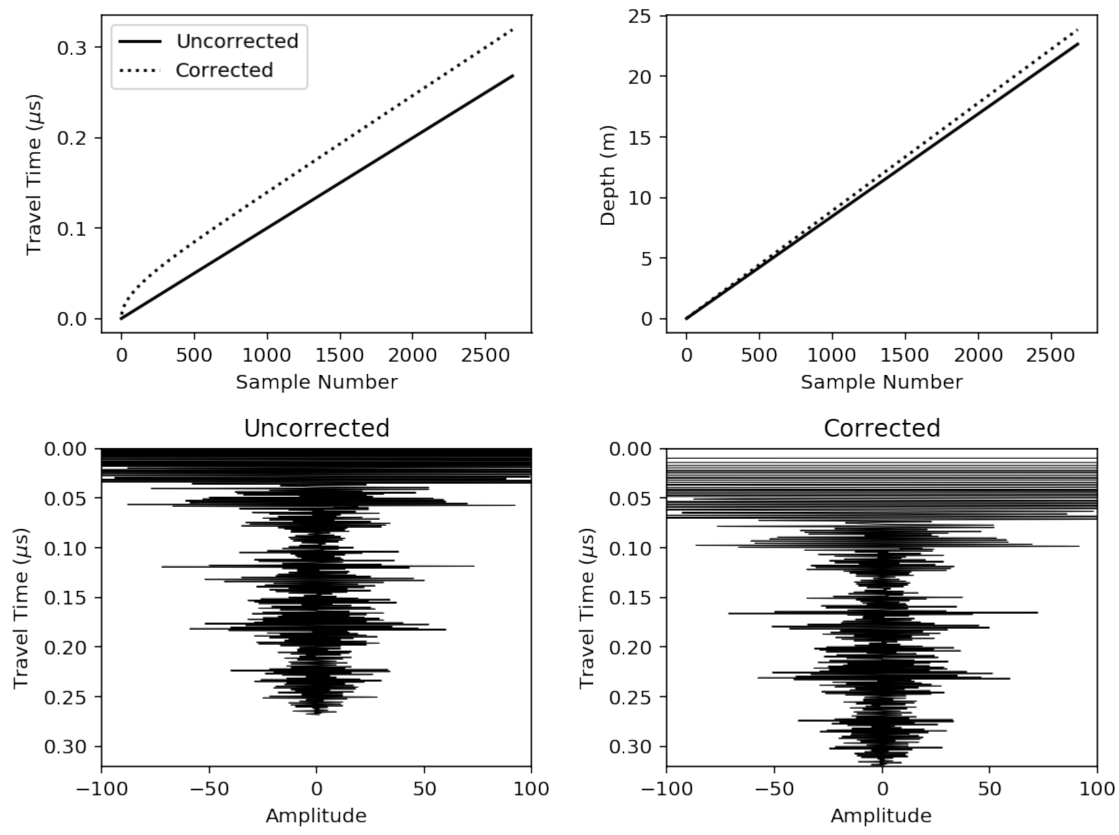
[4]:
```
# Plot a figure before and after the correction
plt.figure(figsize=(8,6))

ax1 = plt.subplot(221)
ax1.plot(dat.travel_time,'k',label='Uncorrected')
plt.xlabel('Sample Number')
plt.ylabel('Travel Time ($\mu$s)')
ax2 = plt.subplot(222)
ax2.plot(dat.travel_time/2.*169,'k')
plt.xlabel('Sample Number')
plt.ylabel('Depth (m)')

ax3 = plt.subplot(223)
plt.plot(dat.data[:,400],dat.travel_time,'k',lw=0.5)
plt.xlim(-100,100)
plt.ylim(0.32,0.0)
plt.ylabel('Travel Time ($\mu$s)')
plt.xlabel('Amplitude')
plt.title('Uncorrected')

# ---------------------

# NMO correction
# 10 m is an overestimate of the antenna separation in this case
# but it is useful to understand what is going on
dat.nmo(10.,rho_profile='2018_density_profile.csv')

# ---------------------

ax1.plot(dat.travel_time,'k:',label='Corrected')
ax1.legend()
ax2.plot(dat.nmo_depth,'k:')

ax4 = plt.subplot(224)
plt.plot(dat.data[:,400],dat.travel_time,'k',lw=0.5)
plt.xlim(-100,100)
plt.ylim(0.32,0.0)
plt.ylabel('Travel Time ($\mu$s)')
plt.xlabel('Amplitude')
plt.title('Corrected')

plt.tight_layout()
plt.show()
```

```
/Users/dlilien/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:3257:␣
↪RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/Users/dlilien/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:161:␣
↪RuntimeWarning: invalid value encountered in double_scalars
```

(continues on next page)

(continued from previous page)

```
  ret = ret.dtype.type(ret / rcount)
/Users/dlilien/work/sw/impdar/impdar/impdar/lib/RadarData/_RadarDataProcessing.py:186:
↪ RuntimeWarning: invalid value encountered in sqrt
  return abs(np.sqrt((t/2.*u_rms)**2.-ant_sep**2.)-d_in)
```



```
[ ]:
```

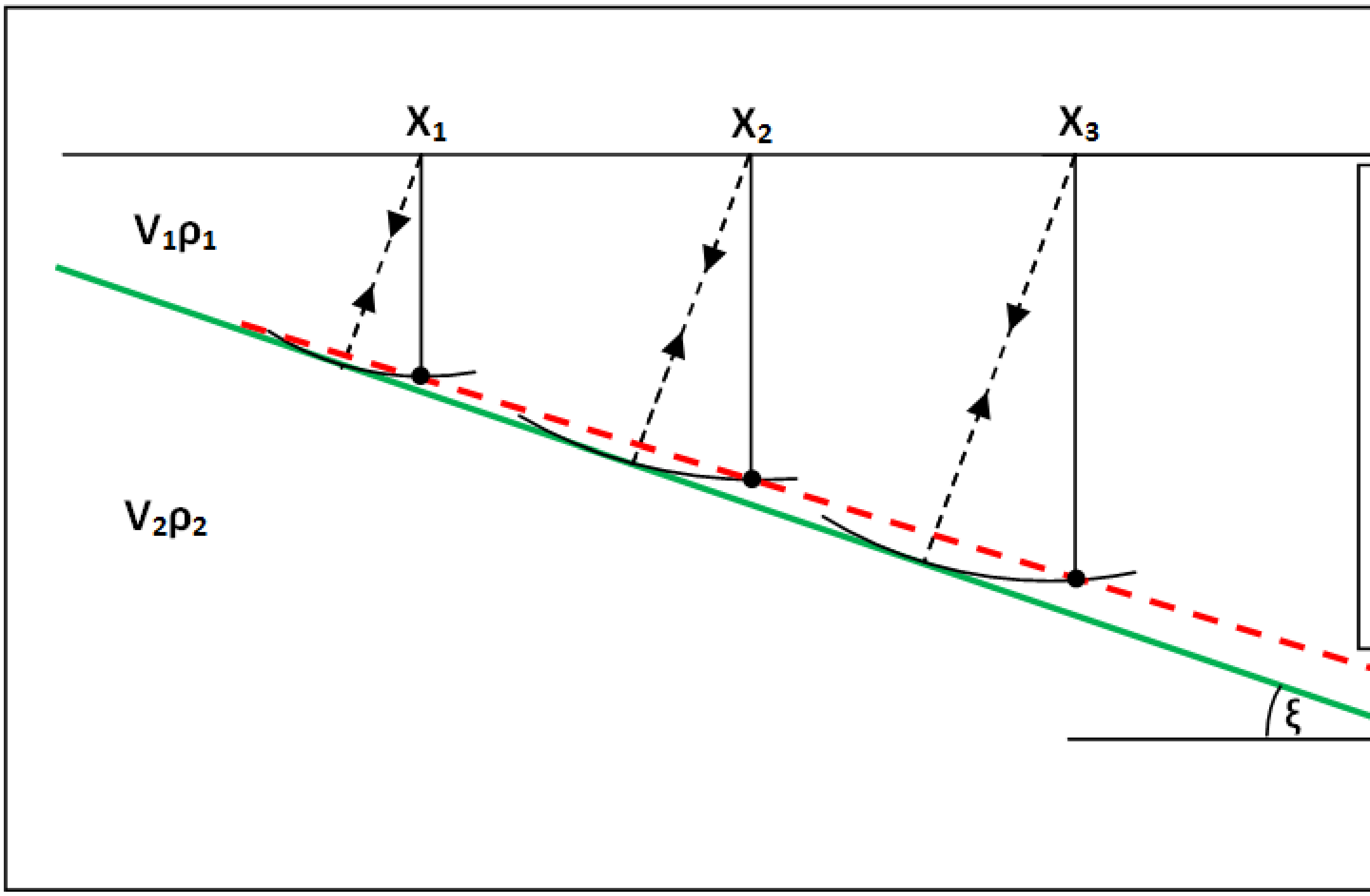
ImpDAR Migration Tutorial

This notebook functions as a sort of tutorial for the migration routines currently implemented in our geophysical processing toolkit for impulse radar (ImpDAR). First, I review a bit of the background on what migration is and why we need to use it to properly interpret geophysical datasets. I then describe the main groups of migration routines by stepping through a synthetic example, migrating the dataset with each method. I discuss the strengths and weaknesses for each as they come up as well as why one method may be chosen over another in a given case. Finally, I implement an actual data example from Hercules Dome.

Much of this material, including the theory but also the algorithms themselves, is taken straight from the textbooks (i.e. Yilmaz, 2001; Sheriff and Geldart, 1995) as well as some of the primary literature (cited in line below).

What is Migration?

The goal of migration is to transform a geophysical dataset into an image that accurately represents the subsurface stratigraphy. Migration is a mathematical transformation in which geophysical events (timing of wave return) are re-located to where the event (the reflection) occurred in the subsurface rather than the location that it was recorded at the surface. Because off-nadir information intrudes into each trace, the image must be migrated as a whole to describe the true reflector geometry. Migration adjusts the angle of dipping reflectors, shortens and moves reflectors updip, unravels bowties, and most generally collapses diffractions.



The migration problem is illustrated in the image above. Dipping reflectors are imaged by an off-nadir ('apparent') reflection. The ray path of the apparent reflection is not the same as the depth of the reflector directly below the source. The *migrator's equation* is a simple analytic way to adjust the angle of a dipping reflector,

$$\tan(\xi_a) = \sin(\xi) \tag{4.1}$$

where ξ is the true reflector dip and ξ_a is the apparent dip shown in the unmigrated image. While this equation is useful, it does not provide the full capability of migrating a full geophysical image. To do that, I explore a few different methods below.

As a note: migration typically assumes coincident source and receiver, meaning that this processing step should be carried out after any stacking or move-out corrections.

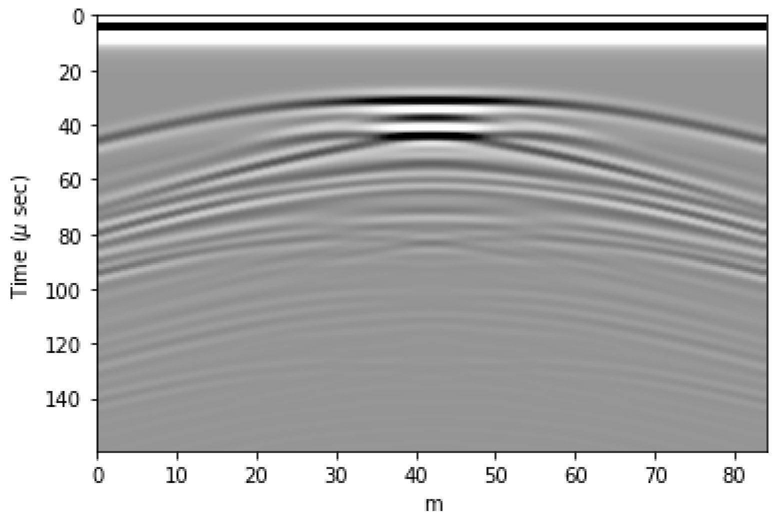
Migration methods outlined below:

- Diffraction Stack Migration (i.e. Kirchhoff)
- Frequency-Wavenumber Migration (e.g. Stolt, Gazdag, etc.)
- Finite-Difference Migration
- SeisUnix Migration Routines (ImpDAR converts to .segy, does the migration in SU, then converts back)

Synthetic Example

Here, I create a synthetic domain to use as an example for the ImpDAR migration routines. For this case, the permittivity is elevated ($\epsilon_r = 12$ inside and 3.2 outside) within the dark blue box in the image below. Loading this domain into gprmax (a finite-difference time-domain modeling software), I simulate a common offset radar survey over the box with the output as a synthetic radargram. The source is a 3-MHz wave from a Hertzian Dipole antenna. Source-receiver antenna separation is 40 m, and the step size between traces is 4 m.

[1]:
```
# We get annoying warnings about backends that are safe to ignore
import warnings
warnings.filterwarnings('ignore')

# Load the synthetic data
from impdar.lib import load
import numpy as np
dat = load.load('mat','data/synthetic_radargram.mat')[0]

# Plot
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(dat.data,cmap='Greys',aspect='auto',vmin=-1,vmax=1,
           extent=[min(dat.dist),max(dat.dist),max(dat.travel_time)*169/2,min(dat.
↪travel_time)*169/2.]);
plt.xlabel('m');
plt.ylabel('Time ($\mu$ sec)');
```



This synthetic image illustrates why we need to migrate. There are large hyperbola that extend away from the actual location of the box in both horizontal directions. These hyperbola, or diffraction curves, do not accurately represent the subsurface stratigraphy, they are only a result of imaging the box from the side as an off-nadir reflector.

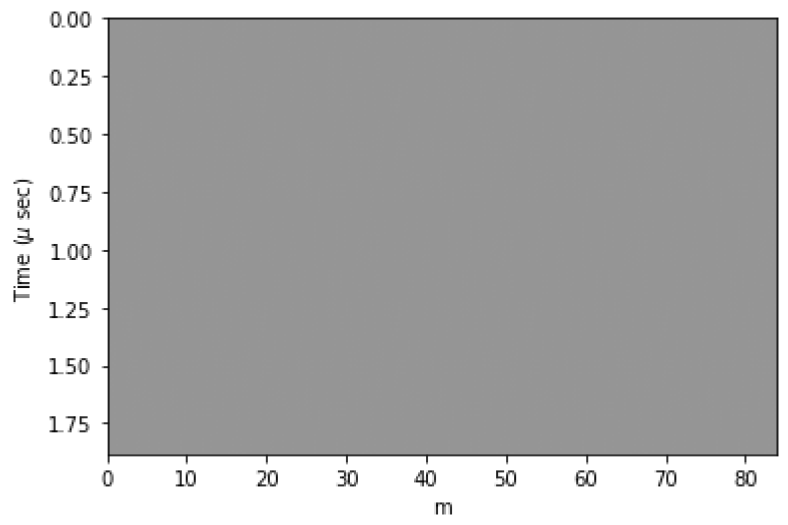
As a note: The domian is slightly different because gprMax needs some room around the edges for a 'perfectly matched layer' boundary condition.

1) Diffraction-Stack Migration – 'Kirchhoff'

The first migration method that I use here is the most direct to explain conceptually. Originally (~1920's), geophysical datasets were migrated by hand, and this method follows the logic used then. The energy along each diffraction curve is summed and placed at the apex of the curve (Hagedoorn, 1954). The diffraction curves are expected to be hyperbolic (in a constant velocity medium they will be), so here we iterate through each point of the image, looking for a hyperbolic diffraction curve and integrating the power along it.

```
[2]: # Reload the unmigrated data
dat = load.load('mat','data/synthetic_radargram.mat')[0]

# Migrate
dat.migrate(mtype='kirch');

# Plot
plt.imshow(dat.data,cmap='Greys',aspect='auto',vmin=-.25,vmax=.25,
           extent=[min(dat.dist),max(dat.dist),max(dat.travel_time),min(dat.travel_
↪time)]);
plt.xlabel('m');
plt.ylabel('Time ($\mu$ sec)');
```

```
Kirchhoff Migration (diffraction summation) of 85x1598 matrix
Kirchhoff Migration of 85x1598 matrix complete in 0.14 seconds
```



Now we can see the box in its original location (i.e. ~200-300 lateral distance and ~0.5-1.0 $\mu$s). This method seems to work, but it is slow (even for this small synthetic dataset) and it 'over migrates' through much of the domain as can be seen by the upward facing hyperbola around the edges and below the box.

Summary of Kirchhoff Migration: - Strengths - Conceptually simple. - Migrates steeply dipping reflectors. - Weaknesses - Slow. - Over migrates. - No lateral velocity variation.

2) Frequency-Wavenumber Migration

Migration is commonly done in the frequency domain. In this case, the transformation is one from vertical frequency (ω_z) to vertical wavenumber (k_z). This transformation is done in the frequency domain, so a 2-D Fourier transform is used in these methods. There are many such migration routines; here I highlight a couple popular ones.

2a) Stolt Migration (Stolt, 1978)

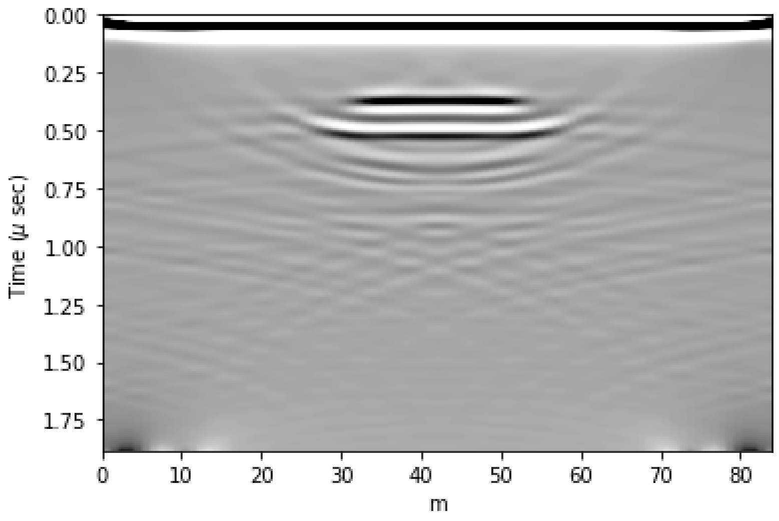
This is the first and probably the simplest of the frequency-wavenumber migration routines. It is done over the entire domain simultaneously, so it requires the assumption of a constant velocity throughout. The transformation done here is

$$P(x,z,t=0) = \int\int \left[\frac{v}{2}\frac{k_z}{\sqrt{k_x^2+k_z^2}}\right] P\left(k_x, 0, v/2\sqrt{k_x^2+k_z^2}\right) e^{-ik_x x - ik_z z} dk_x dk_z$$

where an interpolation is done from ω_z to k_z in frequency-space.

[3]:
```
# Reload the unmigrated data
dat = load.load('mat','data/synthetic_radargram.mat')[0]
twtt = dat.travel_time

# Migrate
dat.migrate(mtype='stolt',htaper=10,vtaper=20);

# Plot
plt.imshow(dat.data,cmap='Greys',aspect='auto',vmin=-1,vmax=1,
           extent=[min(dat.dist),max(dat.dist),max(twtt),min(twtt)]);
plt.xlabel('m');
plt.ylabel('Time ($\mu$ sec)');
```

```
Stolt Migration (f-k migration) of 85x1598 matrix
Interpolating from temporal frequency (ws) to vertical wavenumber (kz):
Interpolating:
0 MHz, 53 MHz, 106 MHz, 159 MHz, 212 MHz, 265 MHz, 318 MHz, 371 MHz, Rescaling TWTT

Stolt Migration of 85x799 matrix complete in 3.12 seconds
```



Stolt migration is great in places where the velocity is known to be constant. It is quite a bit faster than the other routines. Here though, we need to be careful about migrating power in from the edges of the domain, as can be seen in the lower corners above. For this reason, we apply a linear taper to the data so that the Fast Fourier Transform being used does not instantaneously switch from data to zeros around the edges.

Summary of Stolt Migration:

- Strengths
  - Fast.
  - Resolves steeply dipping layers
- Weaknesses
  - Constant velocity.

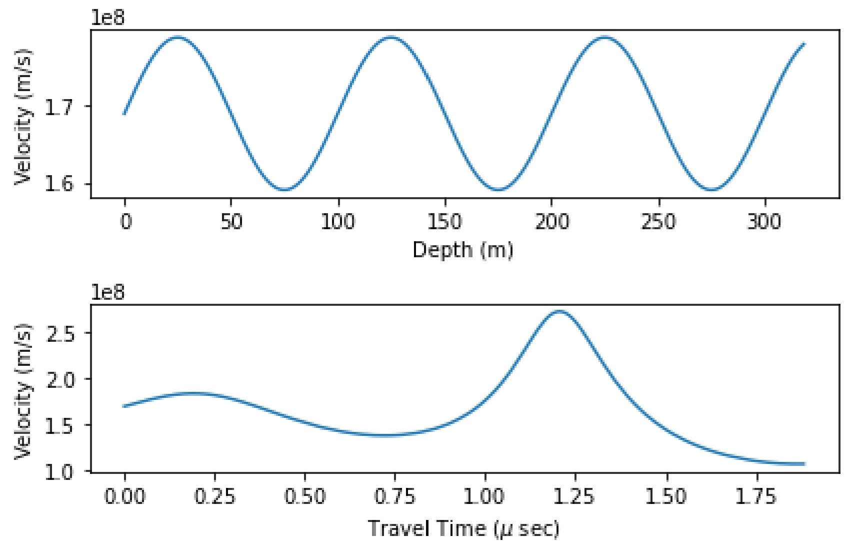
2b) Phase-Shift Migration (Gazdag, 1978)

This second frequency wavenumber migration routines is actually a set of a few. A phase-shifting operator $e^{-ik_z z} is applied at each z-step in downward continuation. These methods are advantageous in that they allow variable velocity as one steps down. Generally, this only allows vertical velocity variation (which I explore here) but there is also a case which accomadates lateral velocity variation (phase-shift plus interpolation) which I will not describe here.

[4]:
```
# Reload the unmigrated data
dat = load.load('mat','data/synthetic_radargram.mat')[0]

from impdar.lib.migrationlib.mig_python import getVelocityProfile
import numpy as np

# create an artificial velocity-depth profile
mean_v = 169e6
z = dat.travel_time*1e-6*mean_v
v = 10e6*np.sin(2.*np.pi*z/100.) + mean_v
vels_in = np.transpose([v,z])

# save velocity profile for migration below
np.savetxt('Phsh_vel_profile.txt',vels_in)

# Use the ImpDAR command to get the interpolated velocity profile
# dependent on travel time instead of depth
vmig = getVelocityProfile(dat,vels_in)
```

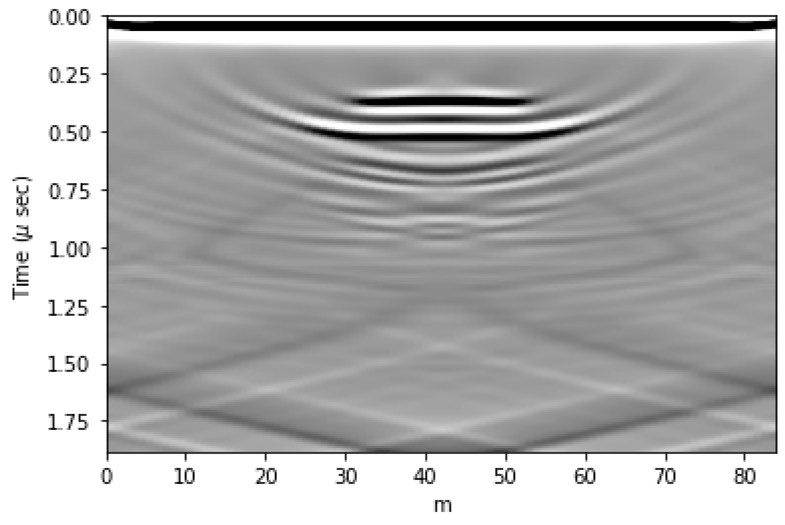
```
Interpolating the velocity profile.
Velocity profile finished in 0.00 seconds.
```

[5]:
```
# Plot
ax1 = plt.subplot(211);
plt.plot(z,v);
plt.ylabel('Velocity (m/s)');
plt.xlabel('Depth (m)');
ax1 = plt.subplot(212);
plt.plot(dat.travel_time,vmig);
plt.ylabel('Velocity (m/s)');
plt.xlabel('Travel Time ($\mu$ sec)');
plt.tight_layout();
```



[6]:
```
# Migrate
dat.migrate(mtype='phsh',htaper=20,vtaper=5,vel_fn='Phsh_vel_profile.txt');

# Plot
plt.imshow(dat.data,cmap='Greys',aspect='auto',vmin=-1,vmax=1,
           extent=[min(dat.dist),max(dat.dist),max(dat.travel_time),min(dat.travel_
↪time)]);
plt.xlabel('m');
plt.ylabel('Time ($\mu$ sec)');
```

```
Phase-Shift Migration of 85x1598 matrix
Velocities loaded from Phsh_vel_profile.txt.
Interpolating the velocity profile.
Velocity profile finished in 0.00 seconds.
1-D velocity structure, Gazdag Migration
Velocities (m/s): %.2e [1.69000000e+08 1.69125225e+08 1.69250429e+08 ... 1.
↪78019586e+08
 1.78072954e+08 1.78124899e+08]
Depths (m): [0.00000000e+00 1.99306291e-01 3.98612583e-01 ... 3.17893535e+02
 3.18092841e+02 3.18292147e+02]
Travel Times ($\mu$ sec): [0.00000000e+00 1.17932717e-03 2.35865434e-03 ... 1.
↪88102683e+00
 1.88220616e+00 1.88338549e+00]
Time 0.00e+00, Time 1.18e-07, Time 2.36e-07, Time 3.54e-07, Time 4.72e-07, Time 5.90e-
↪07, Time 7.08e-07, Time 8.26e-07, Time 9.43e-07, Time 1.06e-06, Time 1.18e-06, Time␣
↪1.30e-06, Time 1.42e-06, Time 1.53e-06, Time 1.65e-06, Time 1.77e-06,
Phase-Shift Migration of 85x1598 matrix complete in 90.70 seconds
```



Much like the result from Kirchhoff migration, we see upward dipping 'smileys' in this migrated image.

Summary of Phase-Shift Migration:

- Strengths
  - Accomodates velocity variation (particularly appropriate for vertical variations, i.e. in snow or similar).
- Weaknesses
  - Maximum dip angle.

3) Finite-Difference Migration

This is a full waveform modeling that essentially runs the arrow of time backward in order to map the geophysical event into an actual reflection location. The beginnings of this method are implemented in ImpDAR, but we have not fully executed this script yet. Because of computational expense, this should probably be written in a high-performance language instead of python. We could do this ourselves, but SeisUnix has already done it (likely better than we would).

4) SeisUnix Migration Routines

There are many migration routines implemented in SeisUnix. I have no desire to replicate the work that they have done, but I wrote something into ImpDAR that allows the user to easily convert their data to .segy, migrate with SeisUnix, then convert back, all in a kind of black box fashion with only one command.

References

Yilmaz (2001). Seismic Data Processing.

Sherrif and Geldart (1995). Exploration Seismology.

Hagedorn (1954). Seismic Imaging Migration.

Stolt (1978). Migration by Fourier Transform.

Gazdag (1978). Wave Equation Migration with the Phase-Shift Method.

ImpDAR ApRES Tutorial

This is an overview of the ApRES functions implemented in ImpDAR. ApRES (Autonomous phase-sensitive Radio Echo Sounder) is a radar system designed to measure vertical ice motion using phase offset (Nicholls et al., 2015). The Python functions in ImpDAR were rewritten from a series of MATLAB scripts (Brennan et al., 2013) (https://discovery.ucl.ac.uk/id/eprint/1425855/1/Brennan_IET-RSN.2013.0053.pdf). The main functionality includes: - loading the data files into an ImpDAR-style .mat file - pulse compression and range conversion - chirp stacking - uncertainty - phase coherence

We overview each of these below.

[1]:
```
# First import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Import the impdar functions that will be needed
import impdar
from impdar.lib.ApresData.load_apres import load_apres
from impdar.lib.ApresData._ApresDataProcessing import apres_range,stacking,
↪phase2range,phase_uncertainty,range_diff

# Plot the data inline instead of with qt5
%matplotlib inline
```

Take a look at the data file

ApRES data files are binary, so we need to read them in with the load_apres function.

[2]:
```
# Load an example file to look at the settings
fname = ['./data/DATA2019-01-01-0337.DAT']
apres_data = load_apres(fname)

print('### File Header ###: \n\n')
for arg in vars(apres_data.header):
    if arg != 'header_string':
        print(arg,': ', vars(apres_data.header)[arg])

print('\n\n\n\n\n\n ### File Data ###: \n\n')
for arg in vars(apres_data):
    print(arg,': ', vars(apres_data)[arg])
```

```
### File Header ###:


fsysclk :  1000000000.0
fs :  40000.0
fn :  ./data/DATA2019-01-01-0337.DAT
file_format :  5
noDwellHigh :  1
noDwellLow :  0
f0 :  199999999.95343387
f_stop :  399999999.90686774
ramp_up_step :  5000.03807246685
ramp_down_step :  5000.03807246685
tstep_up :  2.5e-05
tstep_down :  2.5e-05
```

(continues on next page)

(continued from previous page)

```
snum :  40000
nsteps_DDS :  40000
chirp_length :  1
chirp_grad :  1256646630.09049
nchirp_samples :  40000
ramp_dir :  up
n_attenuators :  1
attenuator1 :  [20]
attenuator2 :  [-4]
tx_ant :  [1]
rx_ant :  [1]
f1 :  400001522.8521079
bandwidth :  200001522.898674
fc :  300000761.4027709
er :  3.18
ci :  168231646.22761327
lambdac :  0.5607707308507499






 ### File Data ###:


snum :  40000
cnum :  100
bnum :  1
data :  [[[0.09815216 1.24504089 1.23912811 ... 1.42715454 1.4263916  1.42673492]
  [1.43409729 1.26480103 1.27113342 ... 1.40258789 1.40041351 1.40285492]
  [1.41525269 1.2676239  1.28131866 ... 1.42040253 1.41937256 1.41994476]
  ...
  [1.42852783 1.23825073 1.23447418 ... 1.41857147 1.41765594 1.41590118]
  [1.42887115 1.26731873 1.27227783 ... 1.42238617 1.42234802 1.42063141]
  [1.4315033  1.25923157 1.27105713 ... 1.41998291 1.42112732 1.42002106]]]
dt :  2.5e-05
decday :  [737436.]
lat :  None
long :  None
chirp_num :  [[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
  48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
  72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
  96 97 98 99]]
chirp_att :  [[20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j 20.-4.j
  20.-4.j]]
```

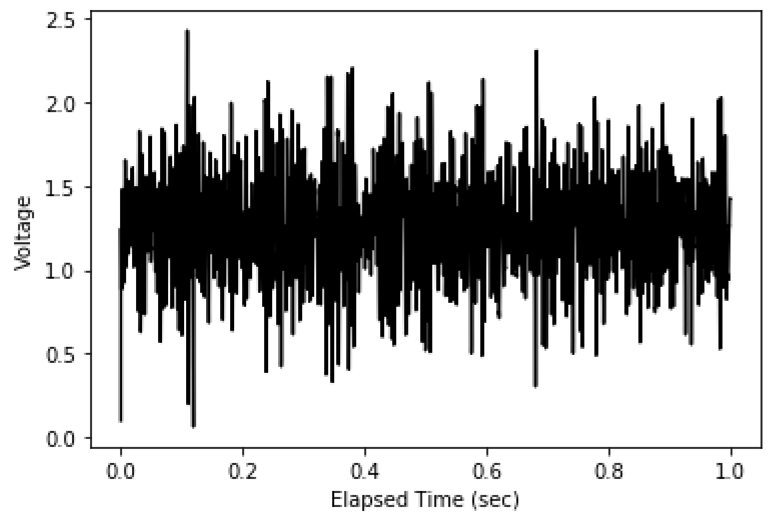
(continues on next page)

(continued from previous page)

```
chirp_time :  [[737435.99998104 737436.         737436.00001896 737436.00003793
  737436.00005689 737436.00007585 737436.00009481 737436.00011378
  737436.00013274 737436.0001517  737436.00017067 737436.00018963
  737436.00020859 737436.00022756 737436.00024652 737436.00026548
  737436.00028444 737436.00030341 737436.00032237 737436.00034133
  737436.0003603  737436.00037926 737436.00039822 737436.00041719
  737436.00043615 737436.00045511 737436.00047407 737436.00049304
  737436.000512   737436.00053096 737436.00054993 737436.00056889
  737436.00058785 737436.00060681 737436.00062578 737436.00064474
  737436.0006637  737436.00068267 737436.00070163 737436.00072059
  737436.00073956 737436.00075852 737436.00077748 737436.00079644
  737436.00081541 737436.00083437 737436.00085333 737436.0008723
  737436.00089126 737436.00091022 737436.00092919 737436.00094815
  737436.00096711 737436.00098607 737436.00100504 737436.001024
  737436.00104296 737436.00106193 737436.00108089 737436.00109985
  737436.00111881 737436.00113778 737436.00115674 737436.0011757
  737436.00119467 737436.00121363 737436.00123259 737436.00125156
  737436.00127052 737436.00128948 737436.00130844 737436.00132741
  737436.00134637 737436.00136533 737436.0013843  737436.00140326
  737436.00142222 737436.00144119 737436.00146015 737436.00147911
  737436.00149807 737436.00151704 737436.001536   737436.00155496
  737436.00157393 737436.00159289 737436.00161185 737436.00163081
  737436.00164978 737436.00166874 737436.0016877  737436.00170667
  737436.00172563 737436.00174459 737436.00176356 737436.00178252
  737436.00180148 737436.00182044 737436.00183941 737436.00185837]]
travel_time :  [0.00000e+00 2.50000e+01 5.00000e+01 ... 9.99925e+05 9.99950e+05
 9.99975e+05]
x_coord :  None
y_coord :  None
elev :  None
flags :  <impdar.lib.ApresData.ApresFlags.ApresFlags object at 0x124670fd0>
header :  <impdar.lib.ApresData.ApresHeader.ApresHeader object at 0x124670e10>
data_dtype :  float64
n_subbursts :  100
average :  0
time_stamp :  [datetime.datetime(2019, 1, 1, 3, 37, 51)]
temperature1 :  [-13.805]
temperature2 :  [-20.656]
battery_voltage :  [0.]
frequencies :  [2.00000000e+08 2.00005000e+08 2.00010000e+08 ... 3.99986523e+08
 3.99991523e+08 3.99996523e+08]
```

The raw volages measured at the receiver are not directly interpretable. However, we plot them below to make it clear exactly what is stored in the data file.

[3]:
```
# plot the raw voltages
plt.figure()
plt.plot(apres_data.travel_time/1e6,apres_data.data[0][0],'k')
plt.ylabel('Voltage')
plt.xlabel('Elapsed Time (sec)');
```

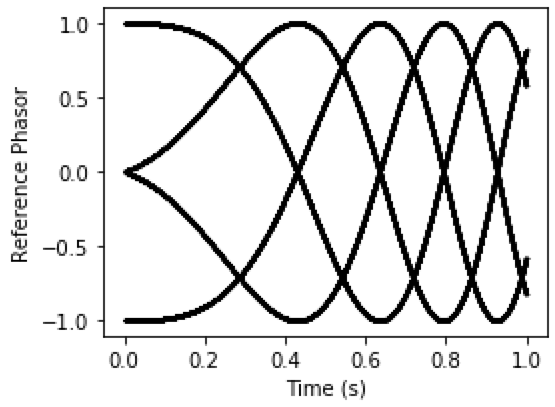


Pulse Compression and Range Conversion

Since this is an FMCW (frequency modulated continuous wave) radar system, the elapsed time in the above figure is not directly convertible to a depth below the surface. Instead, we need to do a pulse compression with the transmitted radar pulse. The code in the next cell are also embedded in the ImpDAR function apres_range(), but we display them here for clarity.

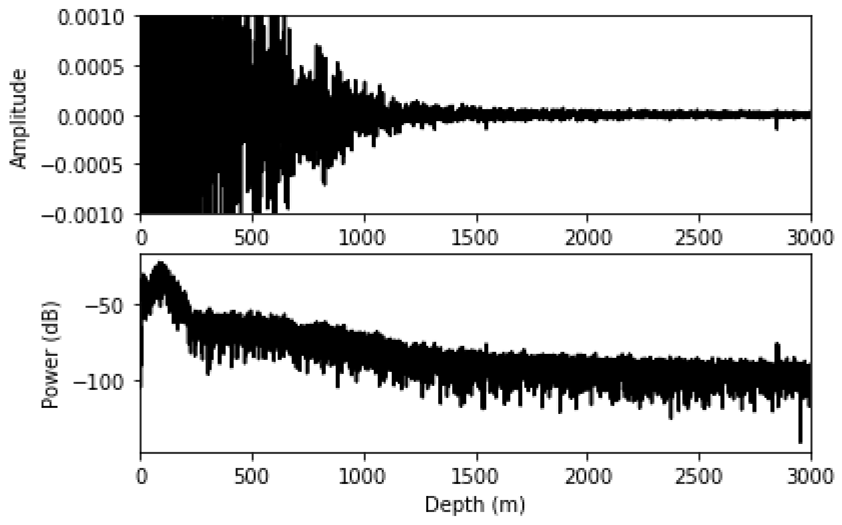
[4]:
```
# Calculate phase for each range bin centre (measured at t=T/2), given that tau = n/
↪(B*p)
t = apres_data.travel_time*1e-6    # Time
K = apres_data.header.chirp_grad   # FM sweep rate
B = apres_data.header.bandwidth    # bandwidth
p = 2  # pad factor for Fourier Transform
nf = int(np.floor(p*apres_data.snum/2))  # number of frequencies to recover
tau = np.arange(nf)/(B*p)  # round-trip delay between antennas
# Brennan et al., (2014) eq 17
_r = 2.*np.pi*apres_data.header.fc*tau - (K*tau**2)/2.  # reference phasor

# ------------------------------------

# Plot the reference phasor
plt.figure(figsize=(4,3))
plt.plot(t,np.exp(-1j*_r),'k.',ms=1)
plt.ylabel('Reference Phasor')
plt.xlabel('Time (s)');
plt.tight_layout()
```

```
/Users/dlilien/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:␣
↪ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



The ImpDAR function, apres_range, uses the reference phasor shown above to convert from the elapsed time to a range measurement. As explaned by Brennan et al. (2013) section 3, the FFT-processed waveform is weighted by the phase conjugate of the reference phasor to do the pulse compression. After executing this function, the data are more interpretable.

[5]:
```
# Reload the data file
apres_data = load_apres(fname)
# pulse compression with the reference phasor
apres_range(apres_data,p)

# Plot one chirp from the data file
plt.figure()
# Amplitude
plt.subplot(211)
plt.plot(apres_data.Rcoarse,apres_data.data[0][0],'k')
plt.xlim(0,3000)
plt.ylim(-.001,.001)
plt.ylabel('Amplitude')
plt.xlabel('Depth (m)')
# Power
plt.subplot(212)
plt.plot(apres_data.Rcoarse,10.*np.log10(apres_data.data[0][0]**2.),'k')
plt.xlim(0,3000)
plt.ylabel('Power (dB)');
plt.xlabel('Depth (m)');
```

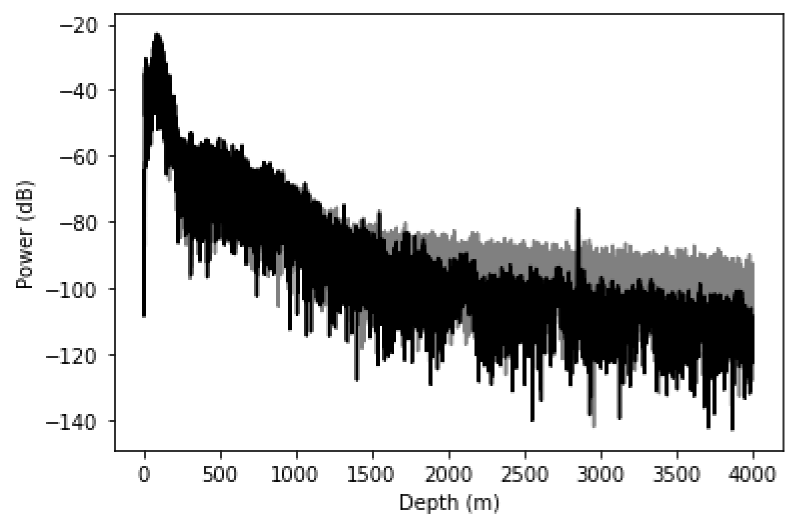


Stacking

As with any system, we want to stack many ApRES chirps together in order to increase the signal-to-noise ratio. The ImpDAR function, stacking(), does this for us by averaging the signal over the given number of chirps.

The ApRES system writes a new file for every 'burst' (set of 100 chirps). However, the load function can handle multiple files if desired. By default, the stacking function will stack across bursts. When stacking each burst individually, change the number of chirps input to the stacking() function to self.cnum.

[6]:
```
# Reload the data file
apres_data = load_apres(fname)
# Pulse compression
apres_range(apres_data,2,max_range=4000)

# Plot the first chirp from the unstacked data
plt.figure()
plt.plot(apres_data.Rcoarse,10.*np.log10(apres_data.data[0][0]**2.),'grey')

# Reload and stack before pulse compression
apres_data = load_apres(fname)
stacking(apres_data)
apres_range(apres_data,2,max_range=4000)

# Plot the stacked chirp
plt.plot(apres_data.Rcoarse,10.*np.log10(apres_data.data[0][0]**2.),'k')
plt.xlabel('Depth (m)');
plt.ylabel('Power (dB)');
```



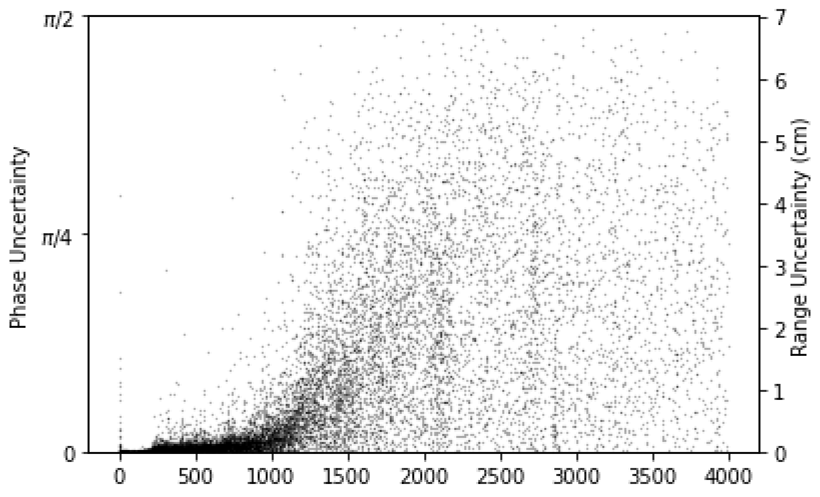
Uncertainty

ImpDAR has two methods for calculating uncertainty in ApRES data. The first is a calculation of the phase uncertainty for a single acquisition using a 'noise phasor' as done by Kingslake et al. (2014). The second is the coherence uncertainty between two acquisitions which we will show later on.

The 'noise phasor' has random phase and amplitude equal to the median amplitude of the measured (or stacked) chirp. This can be calculated with the phase_uncertainty() function in ImpDAR.

[7]:
```
# Reload the data file
apres_data = load_apres(fname)
# Pulse compression
stacking(apres_data)
apres_range(apres_data,2,max_range=4000)
# Calculate the uncertainty
_unc,r_unc = phase_uncertainty(apres_data)

plt.figure()
# phase axis
ax1 = plt.subplot(111)
plt.plot(apres_data.Rcoarse,_unc[0][0],'k.',ms=0.5,alpha=0.5)
plt.ylim(0,np.pi/2.)
plt.yticks([0,np.pi/4.,np.pi/2.],labels=['0','$\pi$/4','$\pi$/2'])
plt.ylabel('Phase Uncertainty');
# twin axis for range
axt = plt.twinx(ax1)
plt.ylim(0,100.*phase2range(np.pi/2.,apres_data.header.lambdac));
plt.ylabel('Range Uncertainty (cm)');
```

```
/Users/dlilien/work/sw/impdar/impdar/impdar/lib/ApresData/_ApresDataProcessing.py:164:
↪ RuntimeWarning: invalid value encountered in arcsin
  p_uncertainty = np.abs(np.arcsin(noise_orth/np.abs(meas_phasor)))
```



Phase Coherence

The coherence between two acquisitions is calculated using a correlation coefficient on samples within a moving window using the function, range_diff(). The output from this function gives: the depths at the center of the window for each step, the coherence resulting from the correlation at each step, the range difference between acquisitions, and the uncertainty.

This coherence value can be used to calculate the range difference (one of the outputs as stated above), but can also be used for polarimetric coherence (e.g. Jordan et al., 2020).

The default uncertainty calculation in this case is to use the Cramer Rao bound (e.g. Jordan et al., 2020). Another option though, is to use the 'noise-phasor' uncertainty as above calculated in each acquisition and added together for the total uncertainty. This is an option in the range_diff() function.

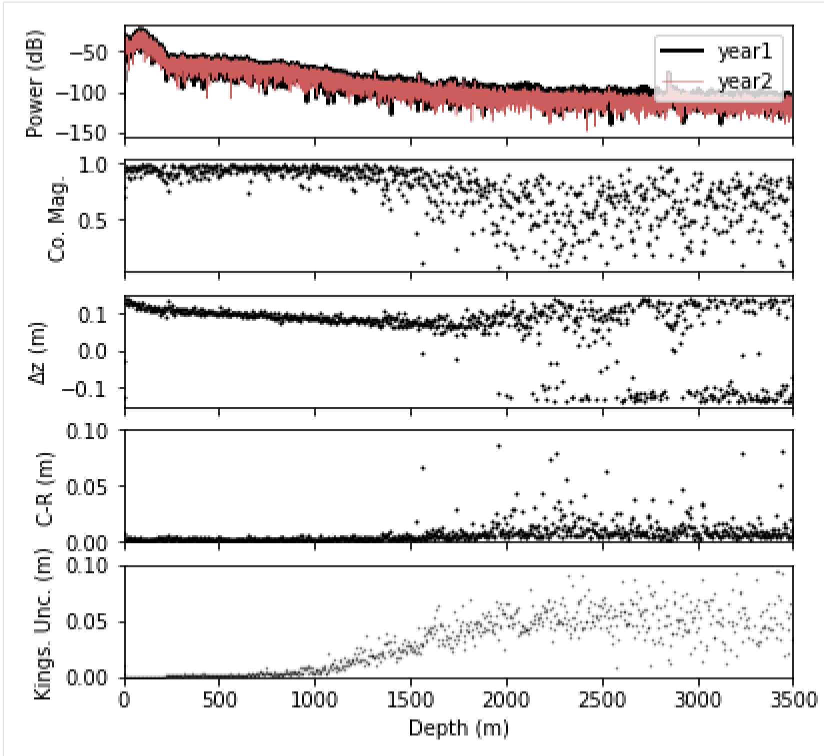
[8]:
```
# Load the data from year 1
fname1 = ['./data/DATA2019-01-01-0337.DAT']
apres_data1 = load_apres(fname1)
stacking(apres_data1)
apres_range(apres_data1,2)
1_unc,r1_unc = phase_uncertainty(apres_data1)
acq1 = apres_data1.data[0][0]
# Load the data from year 2
fname2 = ['./data/DATA2019-12-21-0025.DAT']
apres_data2 = load_apres(fname2)
stacking(apres_data2)
apres_range(apres_data2,2)
2_unc,r2_unc = phase_uncertainty(apres_data2)
acq2 = apres_data2.data[0][0]

# Do the phase-coherence calculation
# over a small moving window from top to bottom of profile
win = 20    # number of samples in the window
step = 20   # window step (number of samples)
ds,phase_diff,r_diff,r_diff_err = range_diff(apres_data1,acq1,acq2,win,step)
amp = abs(phase_diff)  # the magnitude of phase coherence

# ----------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
# Plot a figure
plt.figure(figsize=(6,6))
# Plot the stacked profile from each acquisition
plt.subplot(511)
plt.tick_params(labelbottom=False)
plt.xlim(0,3500)
plt.plot(apres_data1.Rcoarse,10.*np.log10(acq1**2.),'k',lw=2,label='year1')
plt.plot(apres_data2.Rcoarse,10.*np.log10(acq2**2.),'indianred',lw=.5,label='year2')
plt.ylabel('Power (dB)')
plt.legend()
# Plot the magnitude of coherence between seasons
ax2 = plt.subplot(512)
plt.tick_params(labelbottom=False)
plt.xlim(0,3500)
plt.plot(ds,amp,'k.',ms=2)
plt.ylabel('Co. Mag.')
# Plot the phase offset (converted to range)
ax3 = plt.subplot(513)
plt.tick_params(labelbottom=False)
plt.xlim(0,3500)
plt.plot(ds,r_diff,'.',color='k',label='%s'%win,ms=2,zorder=1)
plt.ylabel('$\Delta$z (m)')
# Plot the uncertainty calculated from the Cramer-Rao Bound (Jordan et al., 2020)
plt.subplot(514)
plt.tick_params(labelbottom=False)
plt.xlim(0,3500)
plt.ylim(0,.1)
plt.plot(ds,r_diff_err,'k.',ms=2)
plt.ylabel('C-R (m)');
# Redo the calculation for 'noise-phasor' uncertianty and plot
ds,phase_diff,r_diff,r_diff_err = range_diff(apres_data1,acq1,acq2,win,step,
                                            r_uncertainty=r1_unc[0][0]+r2_unc[0][0],
                                           uncertainty='noise_phasor')
plt.subplot(515)
plt.xlim(0,3500)
plt.ylim(0,.1)
plt.plot(ds,r_diff_err,'k.',ms=1,alpha=0.5)
plt.ylabel('Kings. Unc. (m)');
plt.xlabel('Depth (m)');
```

```
/Users/dlilien/work/sw/impdar/impdar/impdar/lib/ApresData/_ApresDataProcessing.py:289:
↪ RuntimeWarning: Mean of empty slice
  r_diff_unc = np.array([np.nanmean(r_uncertainty[i-win//2:i+win//2]) for i in idxs])
```



ImpDAR plot_power() Tutorial

Introduction

This Notebook shows how, after picking a layer in ice or snow radar data, you would go about verifying that you have picked a single line (and not a combination of multiple lines, meaning you would have to repick that layer). But first, what does it mean to *pick* a layer? Picking is the process of digitizing reflectors within the glacier or ice sheet. As electromagnetic waves are sent through ice or snow, part of that wave is reflected back towards us and can be measured by a receiver antenna. How quickly that electromagnetic wave travels through a given medium is controlled by its permittivity. Each layer that you are seeing in a radargram is the result of permittivity contrasts between layers of different materials like ice, snow, firn, dust, and especially bedrock. Permittivity is heavily dependent on density and conductivity, and so you can clearly see the boundaries between ice-snow, ice-firn, and ice-bedrock where the permittivity changes..

While a pick's power can vary along its length, just as a layer in ice or snow data can vary in depth depending on the surface and bed topography, power should not change drastically. It should be smooth during transitions if it changes at all. If you want to learn more about the way ImpDAR digitizes reflectors, please see the documentation about the `imppick` library here.

Let's take a look at what a picked line looks like when you load it in ImpDAR. As an example, we will use a line collected in early 2020 from Hercules Dome, Antarctica which has already undergone the following processing steps: - the pretrigger data was cropped from the top of the data - a normal move-out correction was applied - the data was vertically bandpassed around a central frequency - then interpolated to regular grid spacing - migrated - and then picked

[1]:
```
# We get annoying warnings about backends that are safe to ignore
import warnings
warnings.filterwarnings('ignore')
```

(continues on next page)

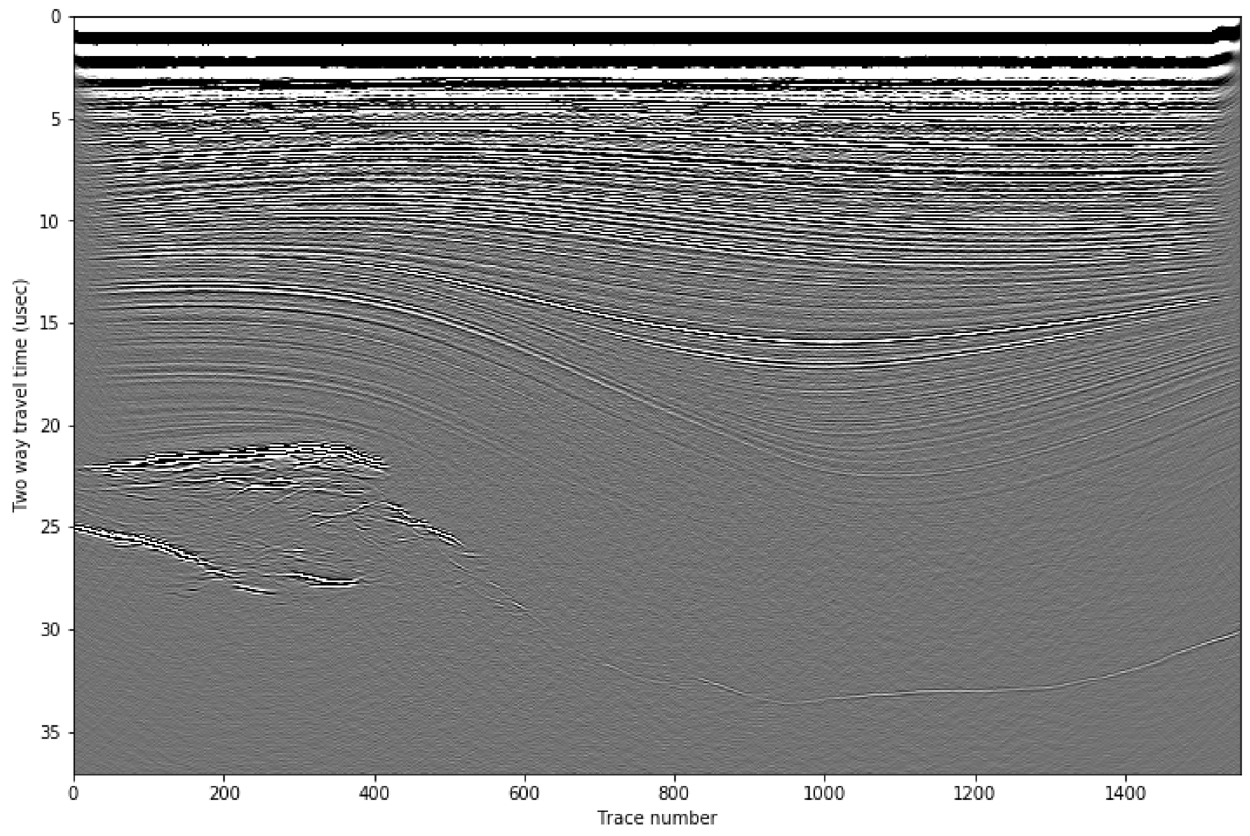
(continued from previous page)

```
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt

import impdar
from impdar.lib import load
from impdar.lib import plot
```

Loading Data

```
[2]: #example matlab file name on disk
herc_mat_file = './data/HDGridE_x53_picks.mat'

#load the hercules dome data, now an ImpDAR RadarData object
dat = load.load('mat', herc_mat_file)[0]
```

```
[3]: #let's inspect our data file
plot.plot_radargram(dat)
plt.show()
```



```
[4]: #we can inspect the RadarData attributes here
vars(dat)
```

```
[4]: {'chan': 2,
      'data': array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
               0.        ,  0.        ],
             [ 0.        ,  0.        ,  0.        , ...,  0.        ,
               0.        ,  0.        ],
             [ 0.        ,  0.        ,  0.        , ...,  0.        ,
               0.        ,  0.        ],
             ...,
             [-0.79655623, -0.78011286, -0.70133412, ..., -0.60073161,
               0.28414989,  0.31941605],
             [-0.47651964, -0.47275442, -0.66581929, ..., -0.68916953,
               0.12462759,  0.28418875],
             [-0.21030229, -0.16901422, -0.61275566, ..., -0.746943  ,
              -0.04463077,  0.22238588]]),
      'decday': array([737454.07234843, 737454.07236893, 737454.07238942, ...,
             737454.10497516, 737454.1049937 , 737454.10501218]),
      'dt': 5e-09,
      'lat': array([-86.44000996, -86.43999383, -86.4399777 , ..., -86.41424587,
             -86.41422667, -86.41420724]),
      'long': array([252.85011635, 252.84942301, 252.84872966, ..., 251.78453531,
             251.78386739, 251.78320123]),
      'pressure': array([0, 0, 0, ..., 0, 0, 0], dtype=uint8),
      'snum': 7415,
      'tnum': 1553,
      'trace_int': array([0, 0, 0, ..., 0, 0, 0], dtype=uint8),
      'trace_num': array([   1,    2,    3, ..., 1551, 1552, 1553], dtype=uint16),
      'travel_time': array([0.0000e+00, 5.0000e-03, 1.0000e-02, ..., 3.7060e+01, 3.
     ↪7065e+01,
             3.7070e+01]),
      'trig': 15,
      'trig_level': array([50, 50, 50, ..., 50, 50, 50], dtype=uint8),
      'nmo_depth': None,
      'elev': array([2595.40362135, 2595.41819483, 2595.43276831, ..., 2598.00668057,
             2597.93407972, 2597.87038998]),
      'dist': array([ 634.3,  639.3,  644.3, ..., 8384.3, 8389.3, 8394.3]),
      'x_coord': array([-369717.25869631, -369717.55360547, -369717.84851463, ...,
             -370193.10233268, -370193.66522139, -370194.2564618 ]),
      'y_coord': array([-114092.32989171, -114097.32118699, -114102.31248227, ...,
             -121824.02863999, -121828.9967592 , -121833.96162683]),
      'fn': '../data/HDGridE_x53_picks.mat',
      'data_dtype': dtype('<f8'),
      'flags': <impdar.lib.RadarFlags.RadarFlags at 0x7fb9a64862e8>,
      'picks': <impdar.lib.Picks.Picks at 0x7fb9a647bcf8>}
```

When we load the data file in ImpDAR, we can access the picks and their corresponding indices with `dat.picks` and `dat.picks.picknums` respectively. `dat.picks.picknums` returns an array back to you with indices that you can use when specifying which picked layer you would like to plot with the `plot_power()` method.

```
[5]: dat.picks.picknums
```

```
[5]: [1, 2]
```

We can also inspect `dat.picks` with the `vars()` method. Otherwise, you will only get back:

```
[6]: dat.picks
```

```
[6]: <impdar.lib.Picks.Picks at 0x7fb9a647bcf8>
```

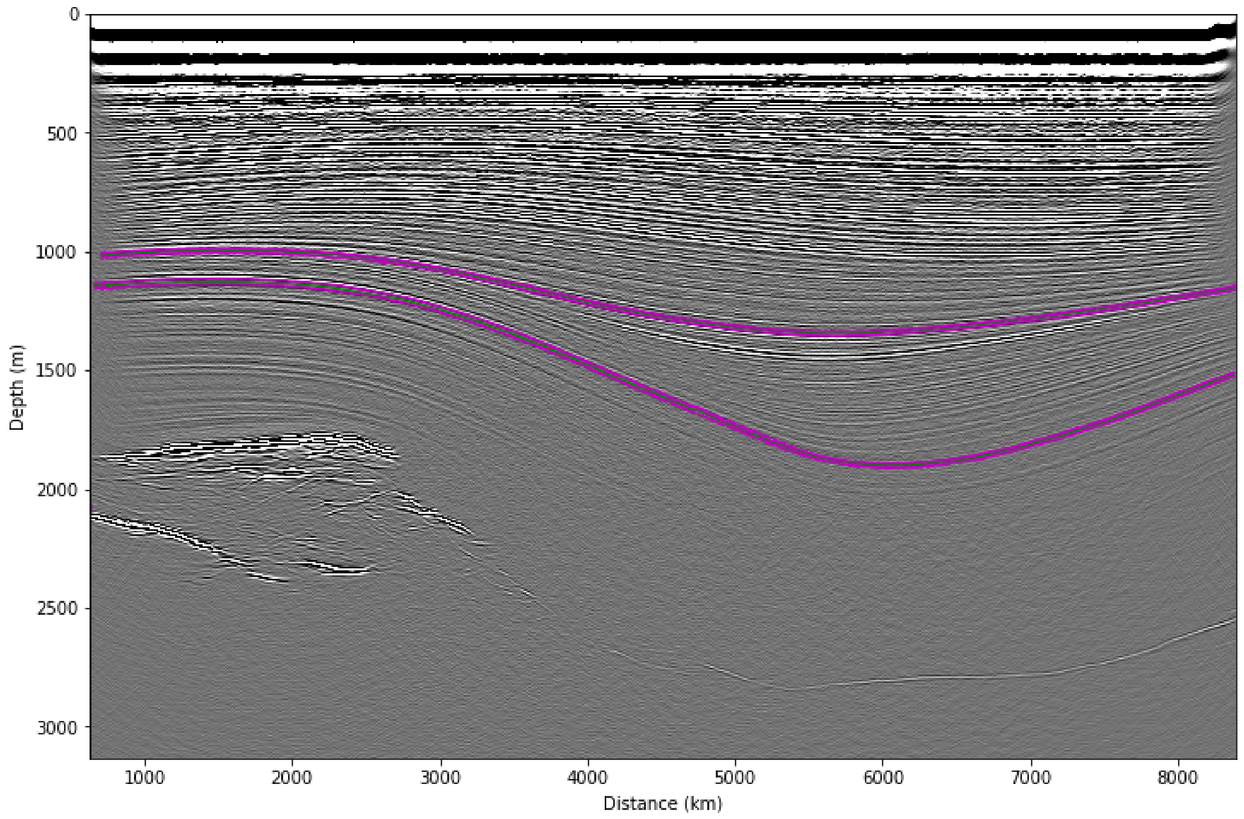
```
[7]: vars(dat.picks)
```

```
[7]: {'samp1': array([[  nan,    nan,    nan, ...,    nan,    nan,    nan],
             [4892., 4892.,    nan, ..., 2711., 2709.,    nan]]),
      'samp2': array([[  nan,    nan,    nan, ...,    nan,    nan,    nan],
             [4925., 4925.,    nan, ..., 2731., 2728.,    nan]]),
      'samp3': array([[  nan,    nan,    nan, ...,    nan,    nan,    nan],
             [4941., 4944.,    nan, ..., 2749., 2746.,    nan]]),
      'time': array([[nan, nan, nan, ..., nan, nan, nan],
             [nan, nan, nan, ..., nan, nan, nan]]),
      'power': array([[        nan,          nan,          nan, ...,          nan,
                      nan,          nan],
             [22.82906175, 30.01928916,          nan, ..., 59.5785543 ,
              45.63542284,          nan]]),
      'picknums': [1, 2],
      'lasttrace': <impdar.lib.LastTrace.LastTrace at 0x7fb9cc33fb70>,
      'lt': <impdar.lib.LeaderTrailer.LeaderTrailer at 0x7fb9cc33fba8>,
      'pickparams': <impdar.lib.PickParameters.PickParameters at 0x7fb99785fcc0>,
      'radardata': <impdar.lib.RadarData.RadarData at 0x7fb9cc33fb38>,
      'lines': []}
```

Plotting Picks

Now we are ready to plot the radargram along with the picks that we have previously saved. ImpDAR's `plot_radargram()` method was designed for this purpose. Note, that without the optional parameters, `plot_radargram()` behaves exactly like the `plot()` method. However, now we can specify our x and y axis units as well as the colors that we would like to use to show our picked lines. Tthe default is magenta-green-magenta for the top, middle, and bottom of the wavelet used to pick lines, and this is what you will encounter in the GUI when you use the `immpick` command from the terminal.

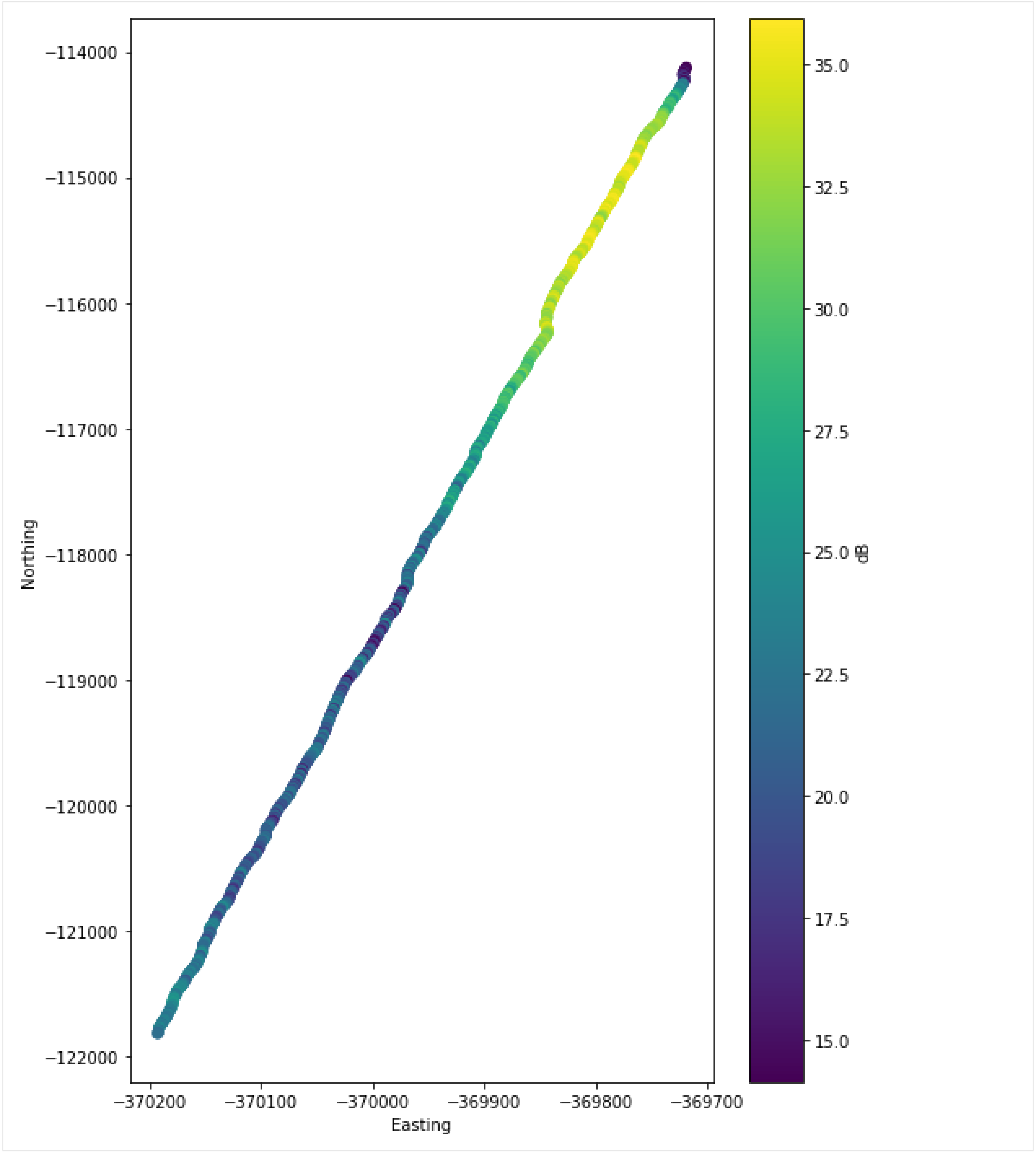
```
[8]: plot.plot_radargram(dat, xdat='dist', ydat='depth', pick_colors='mgm')
     plt.show()
```



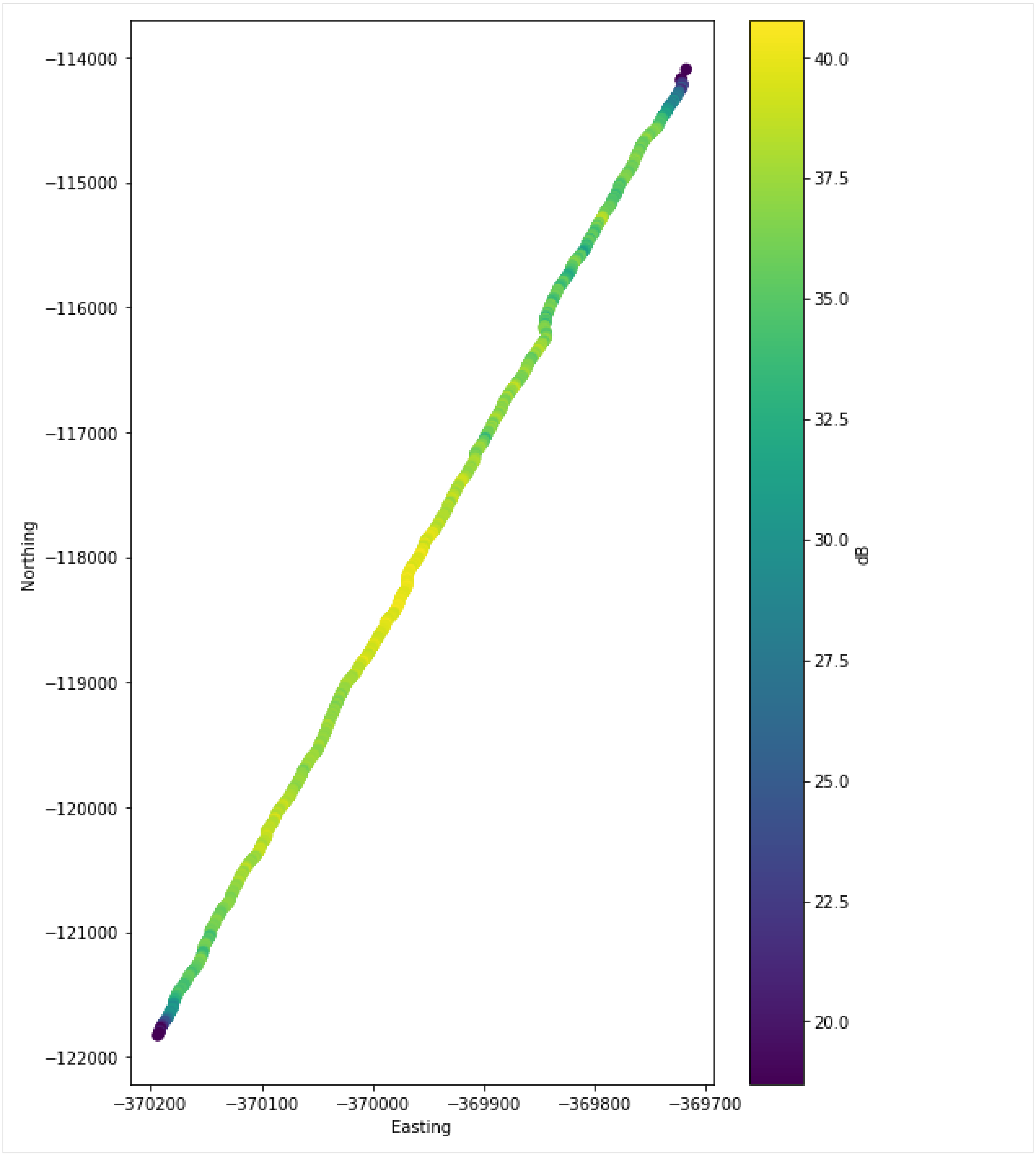
Calculating Power Across Picks

Now we are ready to plot the power of our pick to verify it. We can call the `plot_power()` method from a Jupyter Notebook:

```
[9]: fig, ax = plot.plot_power(dat, 1)
```



```
[10]: fig, ax = plot.plot_power(dat, 2)
```



If you wanted to run this method from the command line instead, you could use the following command to plot the power along the first pick:

```
impdar plot -power 1 HDGridE_x53_picks.mat
```

As you can see, our pick's power does not change drastically along its length. This gives us more confidence that we have correctly identified a single layer within the snow or ice data. Doing this for each and every pick in a radargram profile would be a time-consuming process. But checking important layers at different depths in a radargram's profile can help you quickly verify that you have isolated individual ice and snow layers.

```
[ ]:
```

4.4.2 Additional examples

The primary examples that might be useful are those showing the different *processing* steps and the basics of the *picking gui*.

Additional examples are provided showing *plotting* both via the command line and via the API, and *loading*, though loading in ImpDAR is a single line so the examples are trivial.

Loading Examples

There is not a lot documented here because loading supported files is extremely straightforward in ImpDAR. Loading (i.e. converting raw radar output into the ImpDAR/StoDeep matlab format) is accomplished in a single command with the *impdar load* command.

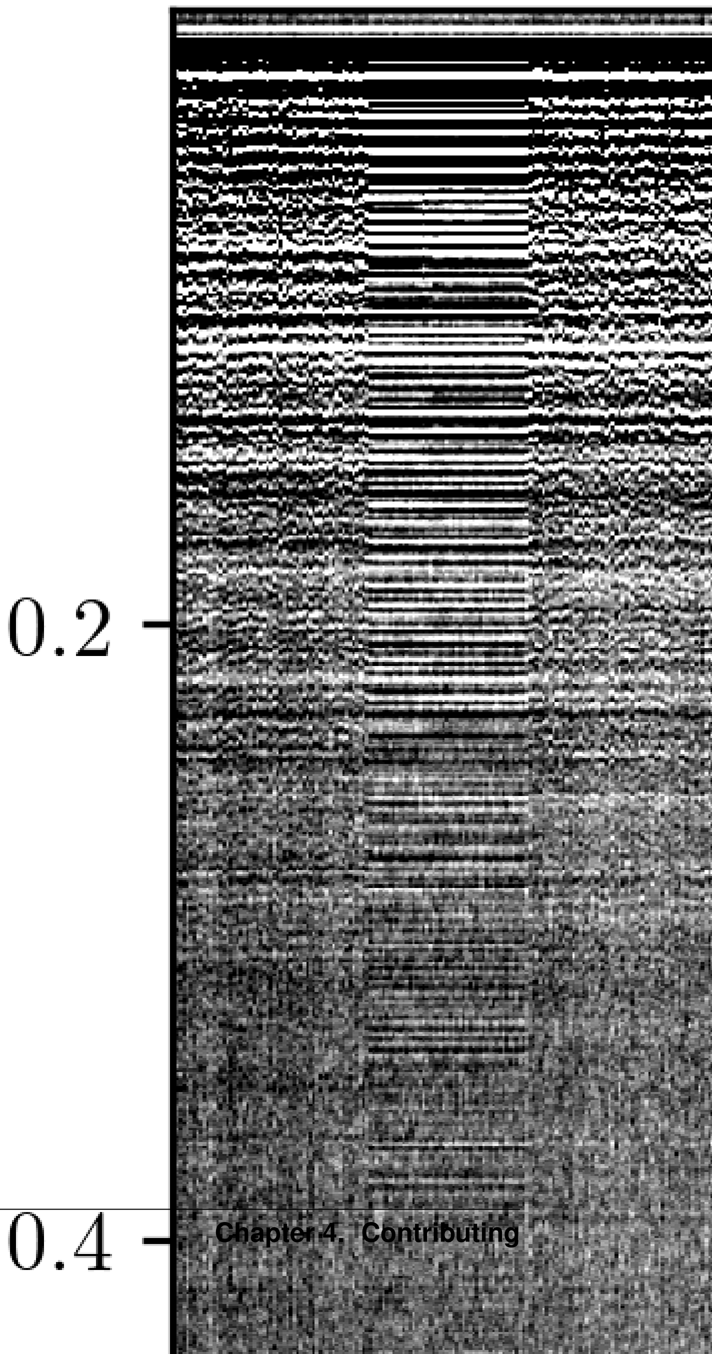
The only real variation amongst filetypes is that you need to tell impdar what type of input file you are using. For example, for GSSI files, `impdar load gssi fn [fn ...]` will produce, for each input fn, a file with identical name but file ext '.mat'. Often, one might want to put all these outputs in a separate folder. The *-o folder_name* allows specification of an output folder. If you wanted to load PulseEkko data, it would be as simple as switching `pe` for `gssi` in the command above.

Processing examples

There are three main options for processing. The first is using `impdar proc` with options in order to do multiple processing steps. `impproc` allows simpler syntax and greater flexibility, but only can apply one processing step at a time. Finally, there are processing options within the pick GUI that allow you to see the effects of the steps immediately, though replotting can be expensive for large datasets and batch processing with the GUI is not possible.

impdar proc

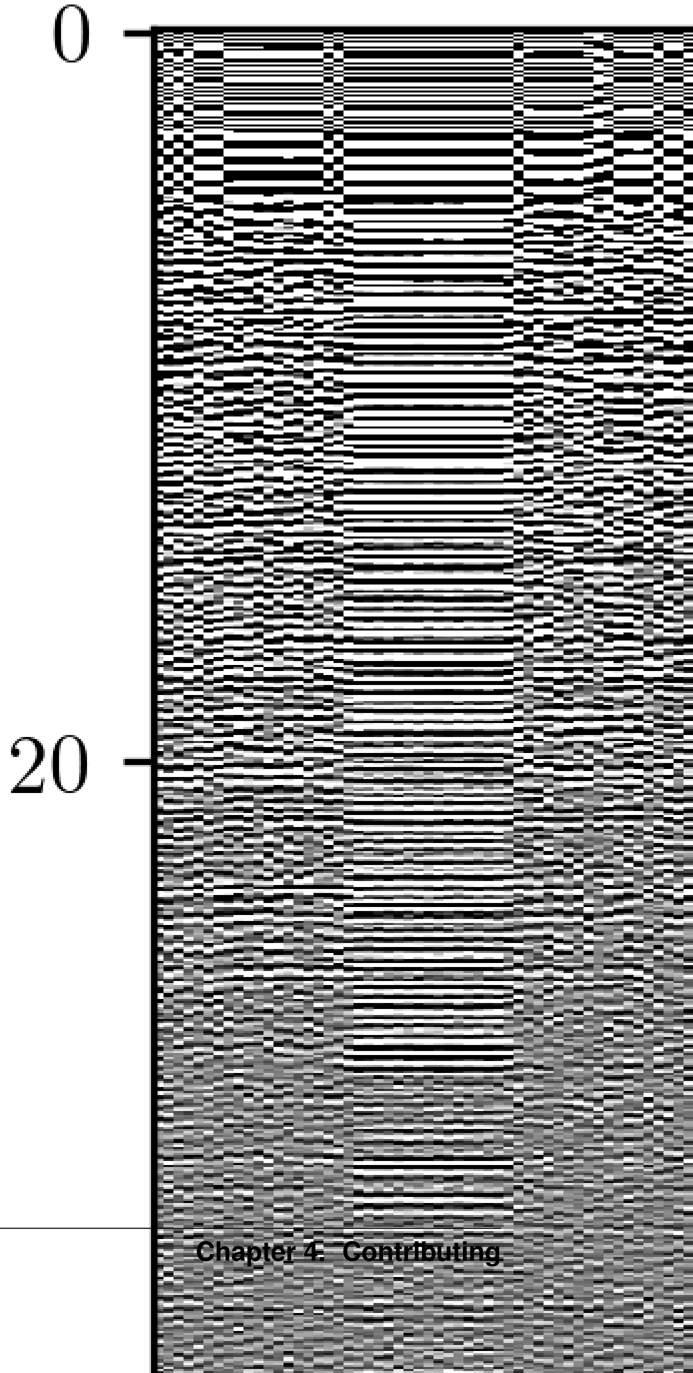
With `impdar proc`, you can perform a number of processing steps in a single line. We are starting with data in crossprofile.mat.


0.2
0.4

This profile does not have anything above the first return; often we would have started recording earlier and have some samples that we would want to delete off the top to start. There is a lot of variability in the overall return power in different traces (resulting from the data collection, not from sub-surface variability). There is also a lot of noise. To vertically bandpass the data between 200 and 600 MHz, adaptively horizontally filter, stack 3 traces, and do a normal moveout correction with no transmit-receive separation only requires running

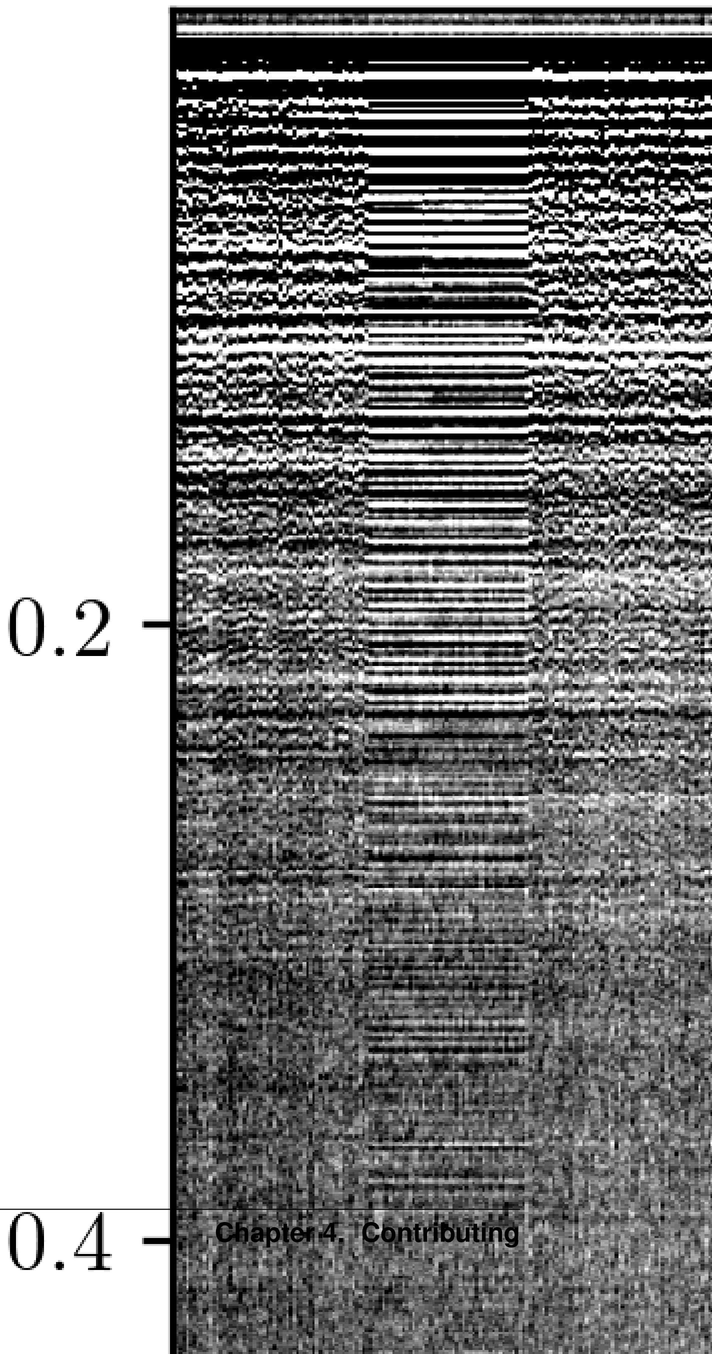
```
impdar proc -vbp 200 600 -ahfilt -restack 3 -nmo 0 1.69 crossprofile.mat
```

and then the output is saved in crossprofile_proc.mat.


0
20

impproc

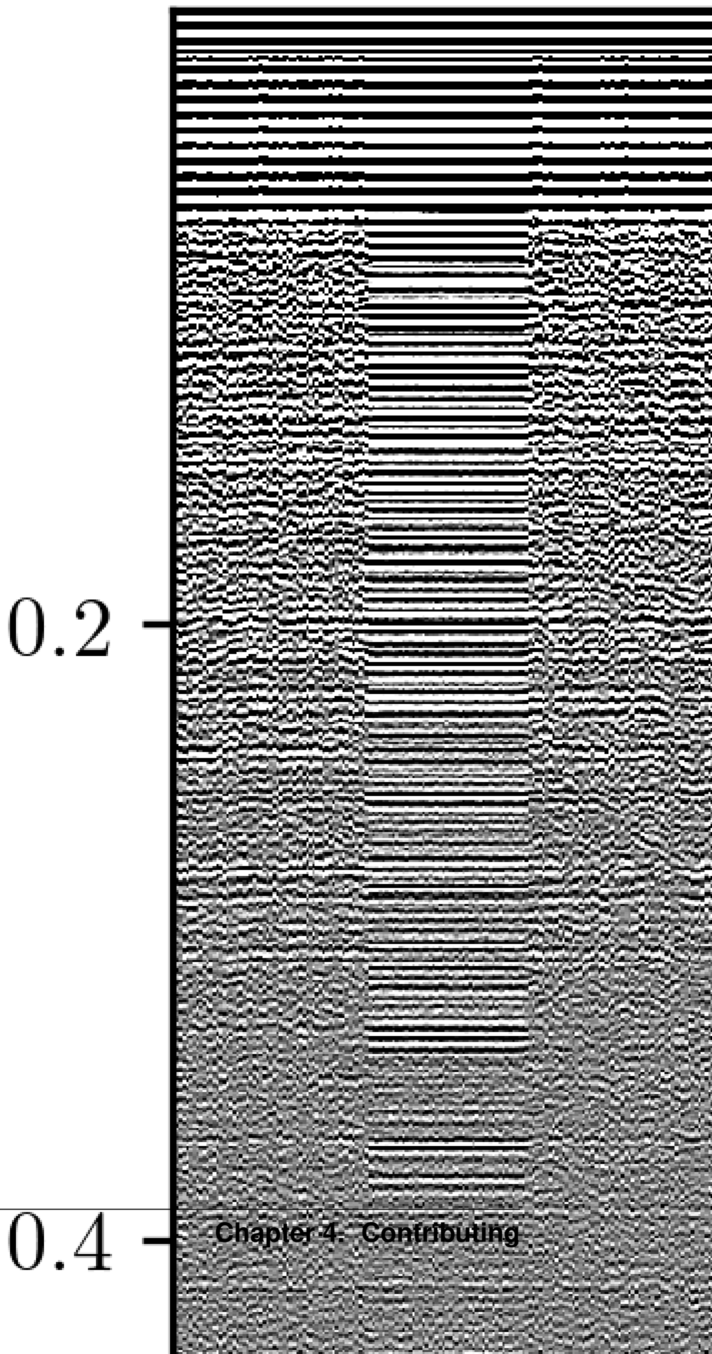
`impproc` provides a bit cleaner syntax than `impdar proc` but accomplishes the same tasks. It is often useful to see the effect of each processing step individually, and `impproc` gives named outputs for each step that allow easy identification and organization. We will use the same example as above, starting with this raw data in crossprofile.mat


0.2
0.4

First, lets do some vertical filtering. As before, we will vertically bandpass with a 5th-order forward-backward Butterworth filter from 200 to 600 MHz.

```
impproc vbp 200 600 crossprofile.mat
```

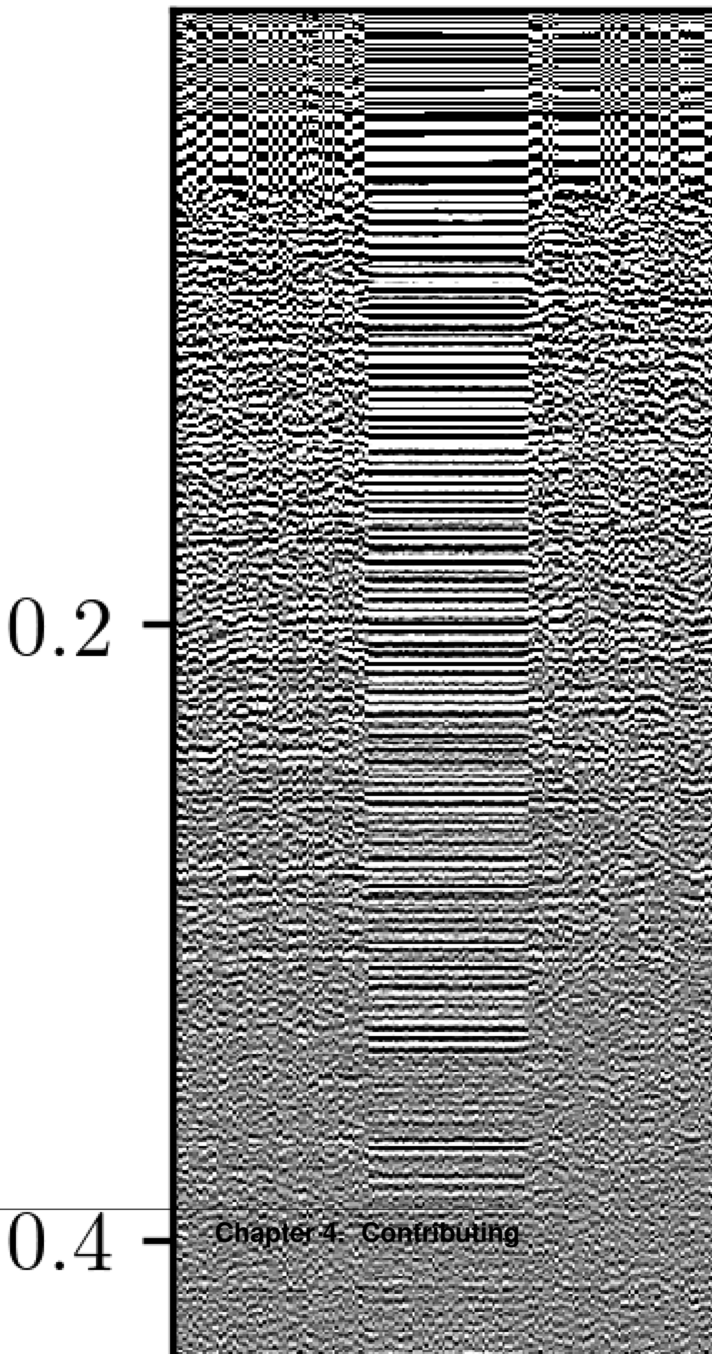
This gives an output in 'crossprofile_bandpassed.mat'. We can see that this has removed most of the noise.


0.2
0.4

We still probably have some noise coming in horizontally (e.g. long-wavelength changes in return power due to our batteries draining in the radar controller). To remove this, we can remove something akin to the average trace.

```
impproc ahfilt crossproile_bandpassed.mat
```

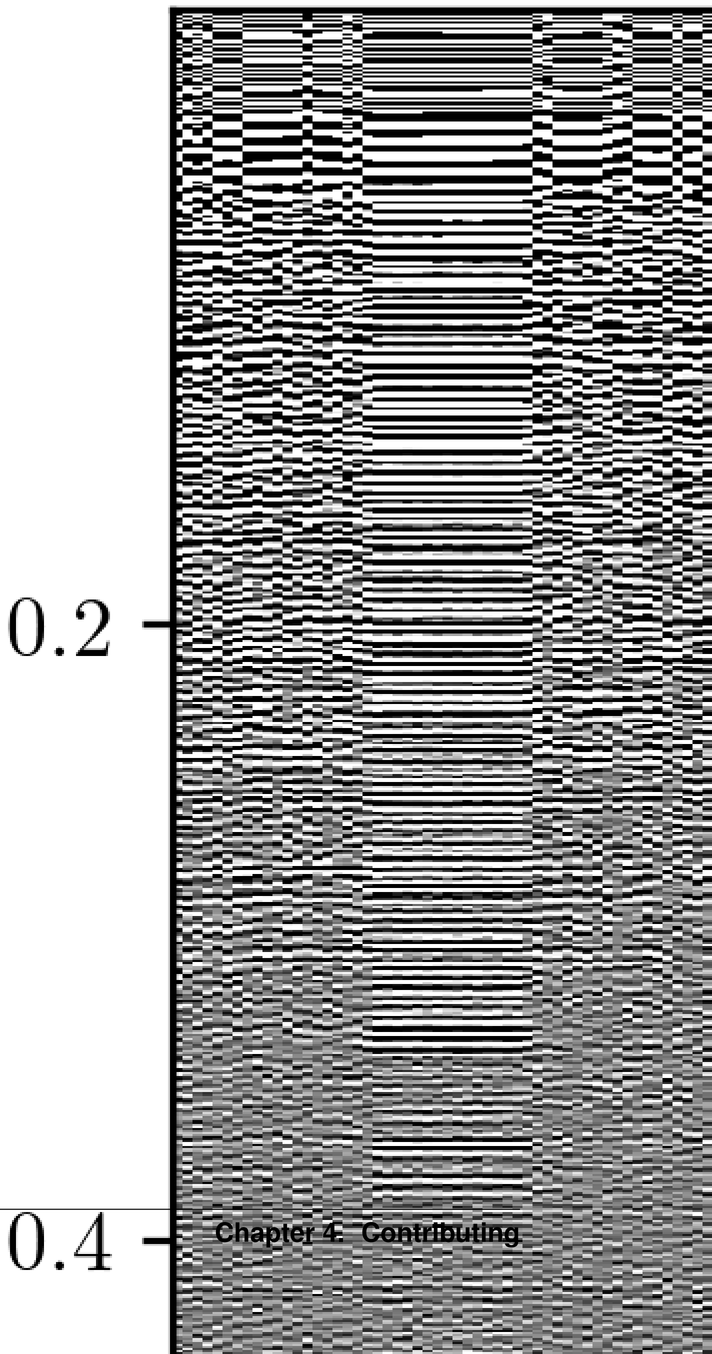
Which gives us 'crossproile_bandpassed_ahfilt.mat'. This looks about the same, though layers have become slightly more clear.


0.2
0.4

Since layer slopes are small, we have lots of extraneous data. We can restack to reduce noise a bit more and reduce filesize.

```
impproc restack 3 crossprofile_bandpassed_ahfilt.mat
```

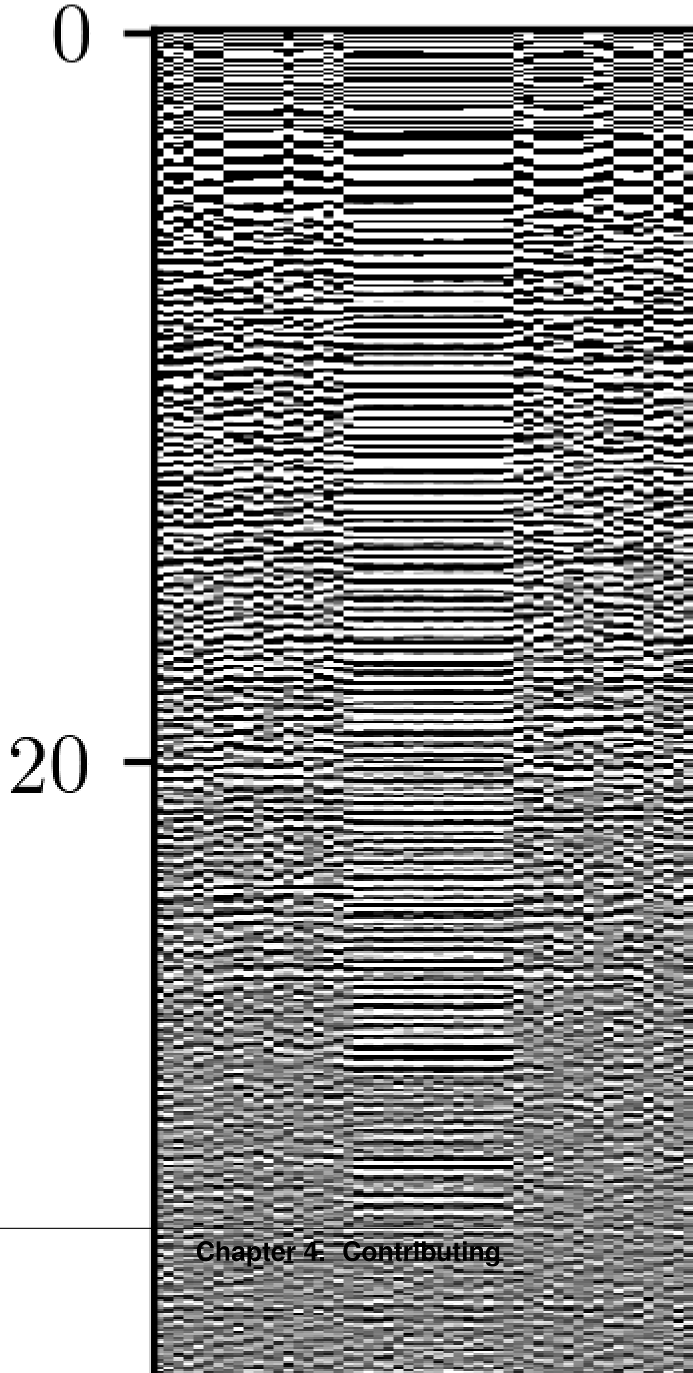
The output is in 'crossprofile_bandpassed_ahfilt_restacked.mat'. Again, this looks about the same, but we have reduced the filesize by about a factor of 3.


0.2
0.4

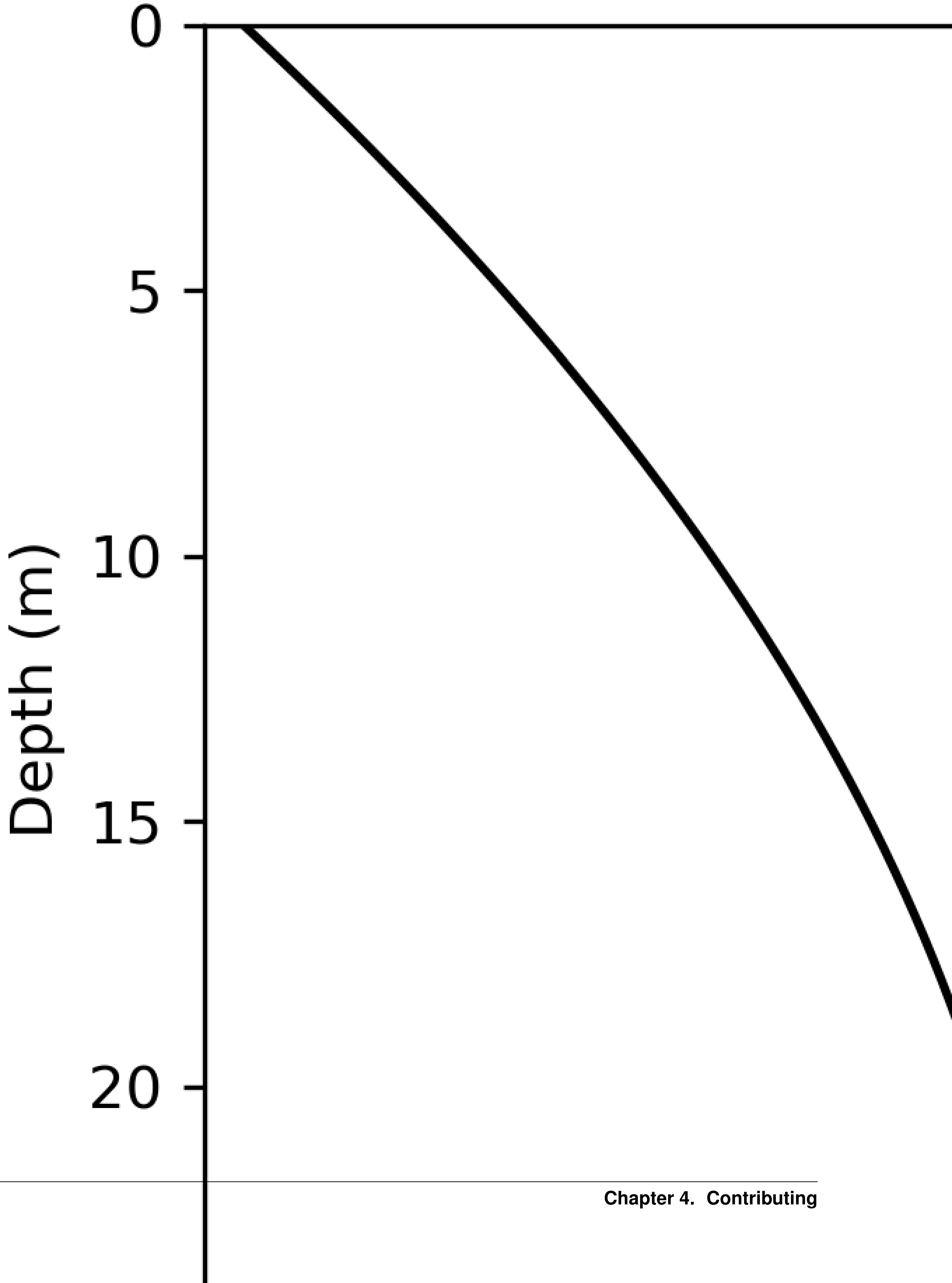
Now we want to look at this in terms of depth. We are going to do this with a constant vertical velocity. This particular data was collected with GSSI radar with a single transmit/receive antenna, so there is no need to do any geometric correction for the triangular pattern of transmit/receive that we would get with spatially separated antennas (like many HF systems have).

```
impproc nmo 0 crossprofile_bandpassed_ahfilt_restacked.mat
```

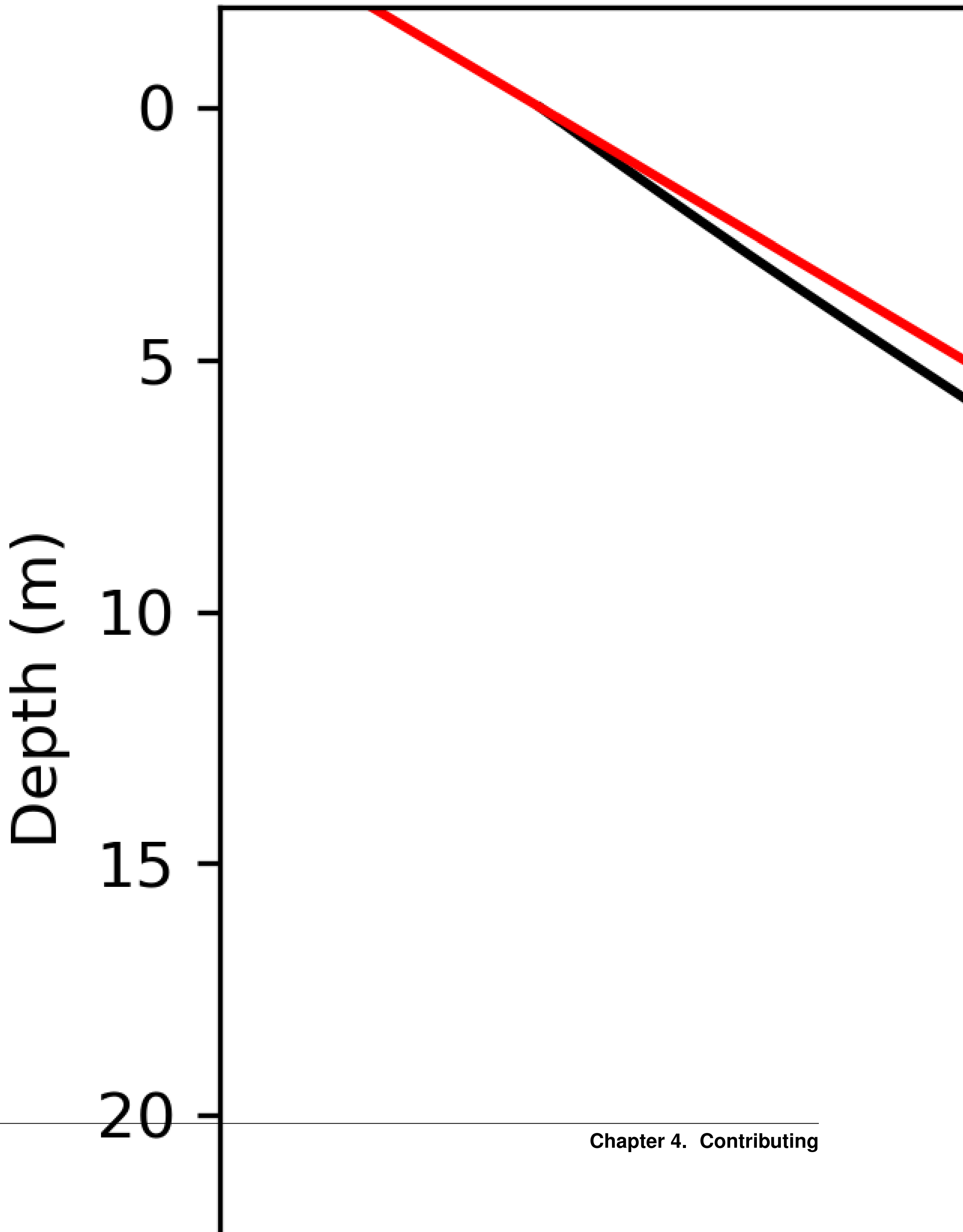
The output is in 'crossprofile_bandpassed_ahfilt_restacked_nmo.mat'. The plot looks identical to before, but we see that the y-axis is now in depth.


0
20

If the permittivity is not constant (for example in the case of variable snow/firn density), we want to make that correction here as well. Optionally, pass a .csv filename as a string to the nmo filter (i.e. rho_profile='__filename__.csv'). The file should have two columns, depth and density. ImpDAR has a couple of options for permittivity models, with the default being the DECOMP mixing model for firn permittivity (Wilhelms, 2005). As an example, here is a measured density profile with modeled permittivity and velocity profiles,


Depth (m)
0
5
10
15
20

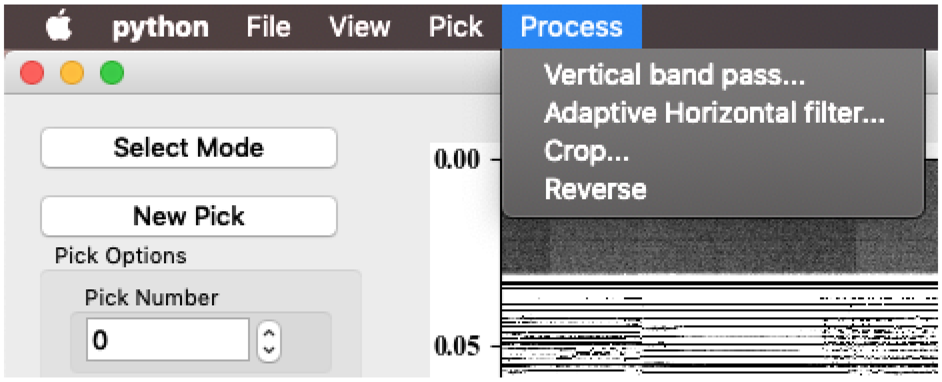
ImpDAR then takes the modeled velocities and updates the depth profile,


Depth (m)
0
5
10
15
20

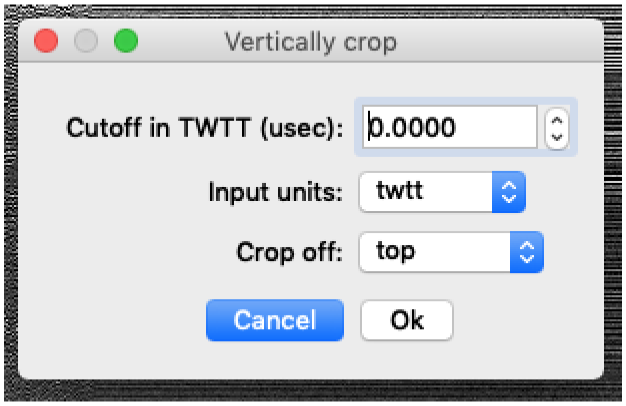
For some datasets, diffraction hyperbolae distort the image, moving much energy away from the true location of the reflecting surface. In these cases, migration is an optional processing step which moves the energy back to its appropriate position in the image. For a more thorough review of the migration routines implemented in ImpDAR, see the next example page on migration.

GUI

After running `imppick`, the GUI has a 'processing' menu.



These options should be self explanatory. If additional arguments are needed by the processing step, a dialog box will be raised. For example, cropping requires information about where you want to crop.



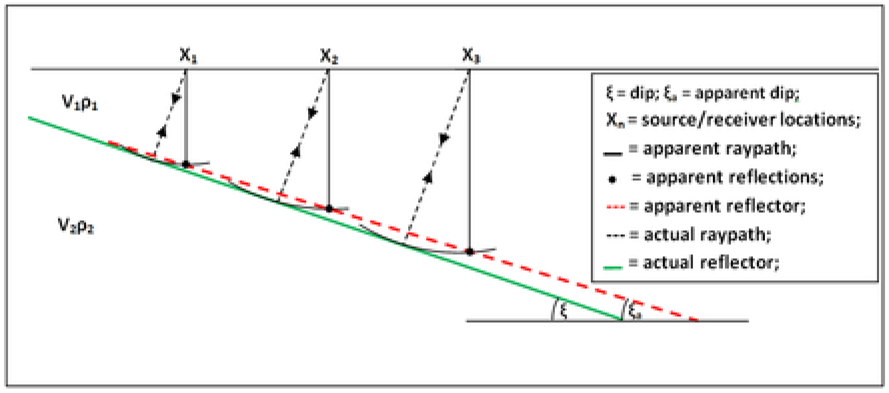
Cutoff in TWTT (usec): 0.0000
Input units: twtt
Crop off: top

There is no automatic saving when processing with the GUI. File > Save (or ctrl command s).

Migration

What is Migration?

The goal of migration is to transform a geophysical dataset (typically seismic data but in this case radar) into an image that accurately represents the subsurface stratigraphy. Migration is a mathematical transformation in which geophysical events (timing of wave return) are re-located to where the event (the reflection) occurred in the subsurface rather than the time at which it was recorded by the receiver at the surface. Because off-nadir information intrudes into each trace, the image must be migrated as a whole to describe the true reflector geometry. Migration adjusts the angle of dipping reflectors, shortens and moves reflectors updip, unravels bowties, and most generally collapses diffractions.


X₁
X₂
X₃
V₁ρ₁
V₂ρ₂
ξ = dip; ξₐ = apparent dip;
Xₙ = source/receiver locations;
— = apparent raypath;
● = apparent reflections;
--- = apparent reflector;
--- = actual raypath;
— = actual reflector;
ξ
ξₐ

The migration problem is illustrated in the image above. Dipping reflectors are imaged by an off-nadir ('apparent') reflection. The ray path of the apparent reflection is not the same as the depth of the reflector directly below the source. The migrator's equation is a simple analytic way to adjust the angle of a dipping reflector,

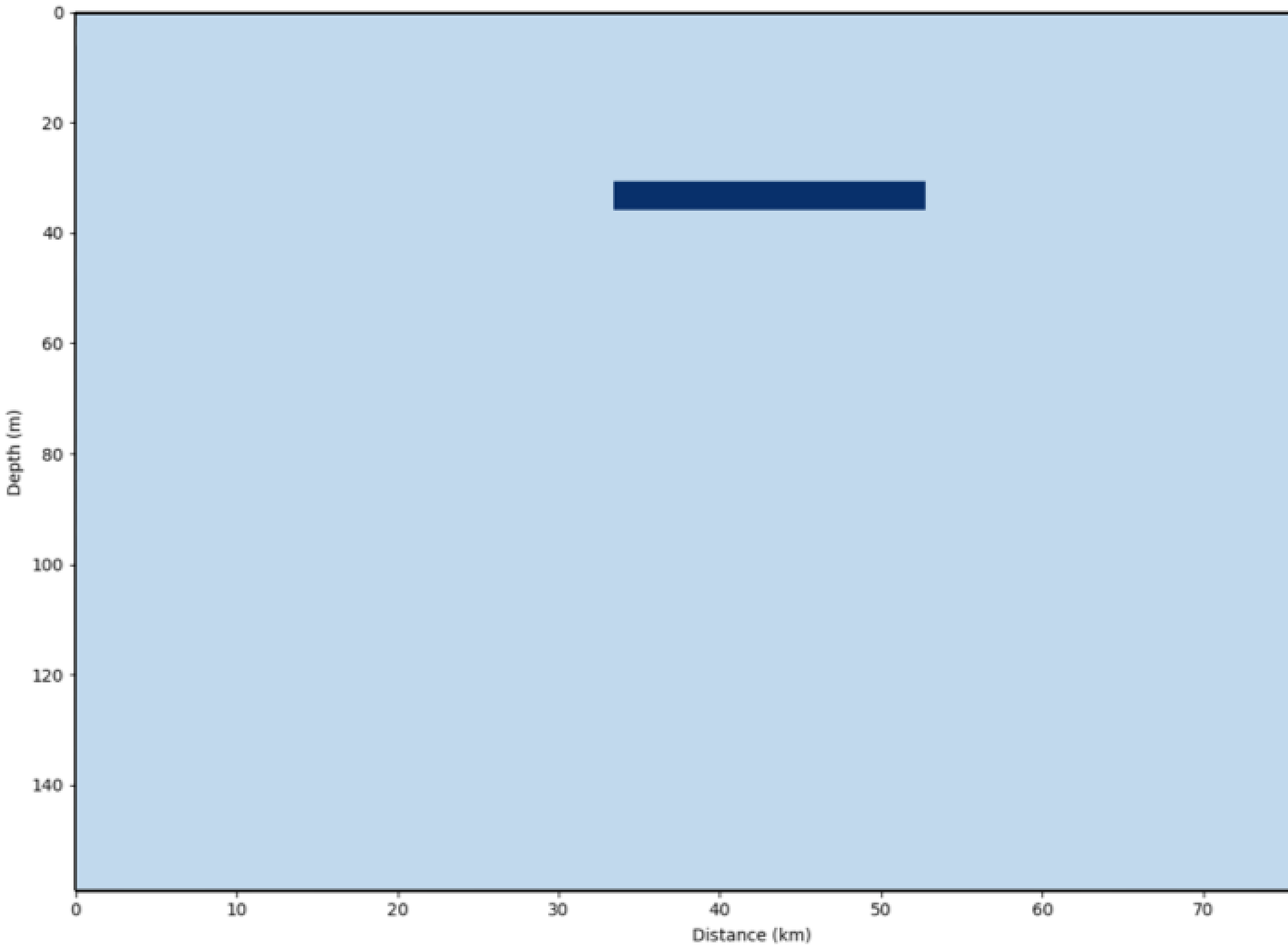
$$tan(\xi_a) = sin(\xi)$$

where ξ is the true reflector dip and ξ_a is the apparent dip shown in the unmigrated image. While this equation is useful, it does not provide the full capability of migrating the entire image. To do that, we explore a few different methods below.

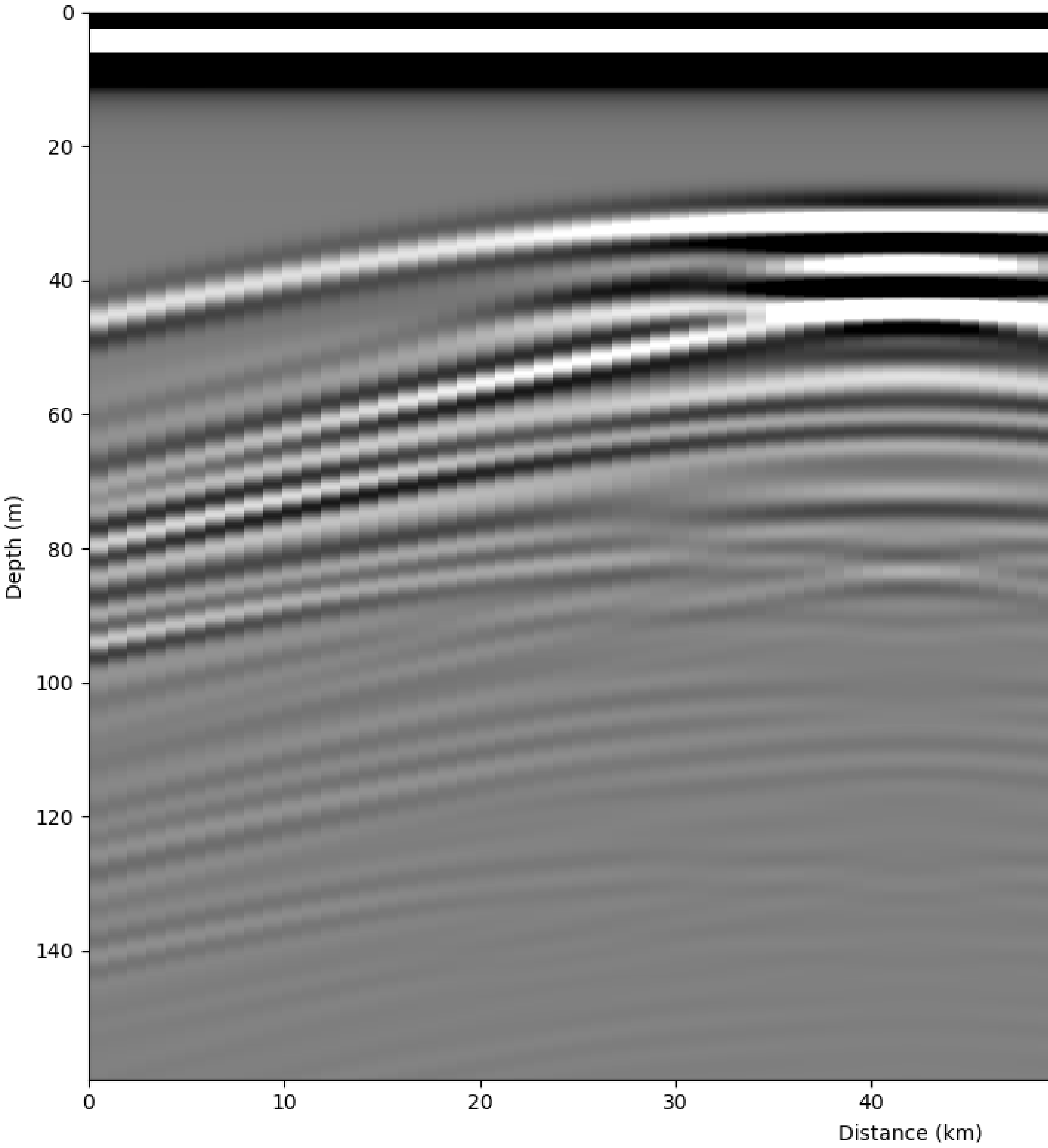
Note: migration typically assumes coincident source and receiver, meaning that this processing step should be carried out after any stacking or move-out (nmo) corrections.

Synthetic Example

Here, we create a synthetic domain to use as an example for the ImpDAR migration routines. For this case, the permittivity is elevated within the dark blue box in the image below ($\epsilon_r = 12$ inside and 3.2 for ice outside).



Loading this domain into gprmax (a finite-difference time-domain modeling software), we simulate a common-offset radar survey over the box with the output as a synthetic radargram. The source is a 3-MHz wave from a Hertzian Dipole antenna. Source-receiver antenna separation is 40 m, and the step size between traces is 4 m.

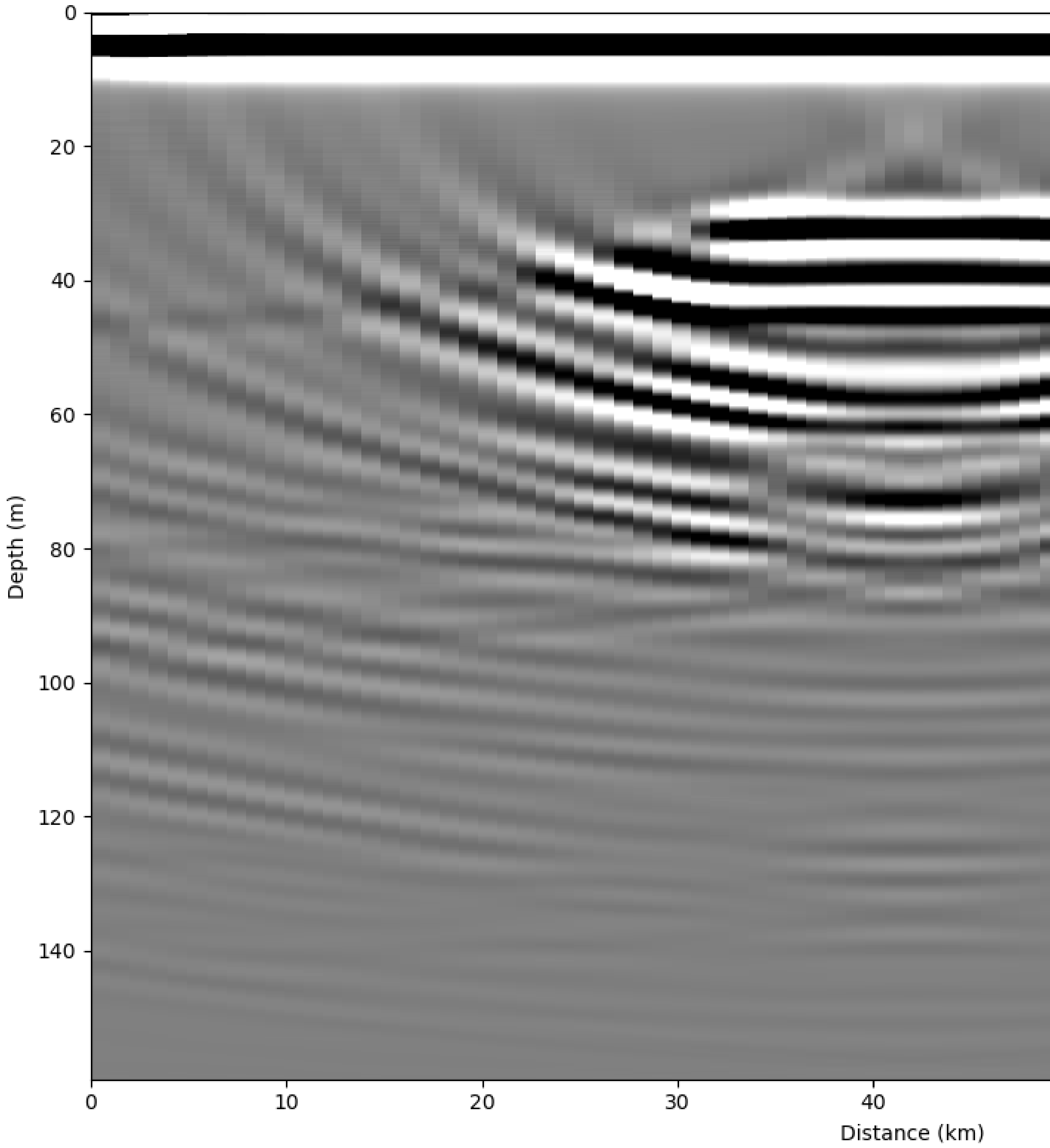


This synthetic image illustrates why we need to migrate. There are large hyperbolae that extend away from the actual location of the box in both horizontal directions. These hyperbola, or diffraction curves, do not accurately represent the subsurface stratigraphy, they are only a result of imaging the box from the side as an off-nadir reflector.

Kirchhoff Migration

The first migration method that we use here is the most direct to explain conceptually. Originally (~1920's), geophysical datasets were migrated by hand, and this method follows the logic used then. The energy is integrated along each diffraction curve and placed at the apex of the curve (Hagedoorn, 1954). The diffraction curves are expected to be hyperbolic (in a constant velocity medium they will be), so here we iterate through each point of the image, looking for a hyperbolic diffraction curve around that point and integrating the power along it.

```
impdar migrate --mtype kirch synthetic.mat
```



Now we can see the box in its original location (i.e. ~30-55 km lateral distance and ~30 m depth). This method seems to work, but it is slow (even for this small synthetic dataset) and it ‘over migrates’ through much of the domain as can be seen by the upward facing hyperbola (‘smileys’) around the edges and below the box.

Summary of Kirchhoff Migration:

- Strengths - Conceptually simple, Migrates steeply dipping reflectors.
- Weaknesses - Slow, Over migrates, No lateral velocity variation.

Stolt Migration

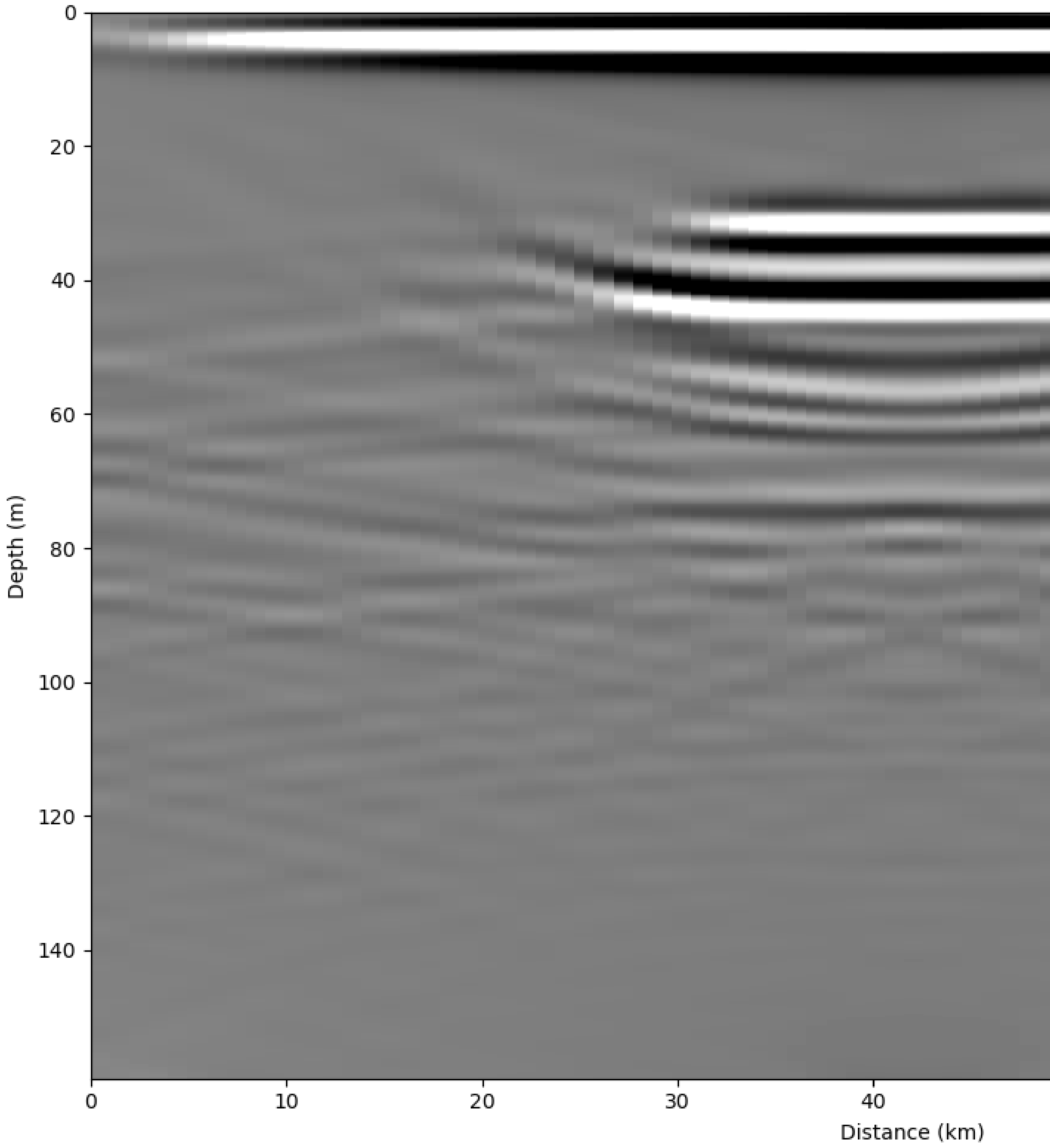
Migration is most commonly done in the frequency domain. In this case, the transformation is one from vertical frequency (ω_z) to vertical wavenumber (k_z); thus, these migration routines are grouped as 'frequency-wavenumber' routines. The transformation is done in the frequency domain, so a 2-D Fourier transform is used before the migration and an inverse Fourier transform after. There are many such migration routines; here I highlight a couple popular ones which have been implemented in ImpDAR.

The first, and probably the simplest, of the frequency-wavenumber migration routines is 'Stolt Migration'. Stolt Migration is done over the entire domain simultaneously, so it requires the assumption of a constant velocity throughout. The transformation is

$$P(x,z,t=0) = \int\int \left[\frac{v}{2}\frac{k_z}{\sqrt{k_x^2+k_z^2}}\right] P\left(k_x, 0, \frac{v}{2}\sqrt{k_x^2+k_z^2}\right) e^{ik_x x ik_z z} dk_x dk_z$$

where an interpolation is done from ω_z to k_z in frequency-space. The routine is implemented in ImpDAR as,

```
impdar migrate --mtype stolt synthetic.mat
```



Stolt migration is great in places where the velocity is known to be constant. It is quite a bit faster than the other routines. Here though, we need to be careful about migrating power in from the edges of the domain, as can be seen in the lower corners above. For this reason, we apply a linear taper to the data so that the Fast Fourier Transform has a smooth transition from data to the zeros that it fills in around the edges.

Summary of Stolt Migration:

- Strengths – Fast, Resolves steeply dipping layers.
- Weaknesses – Constant velocity.

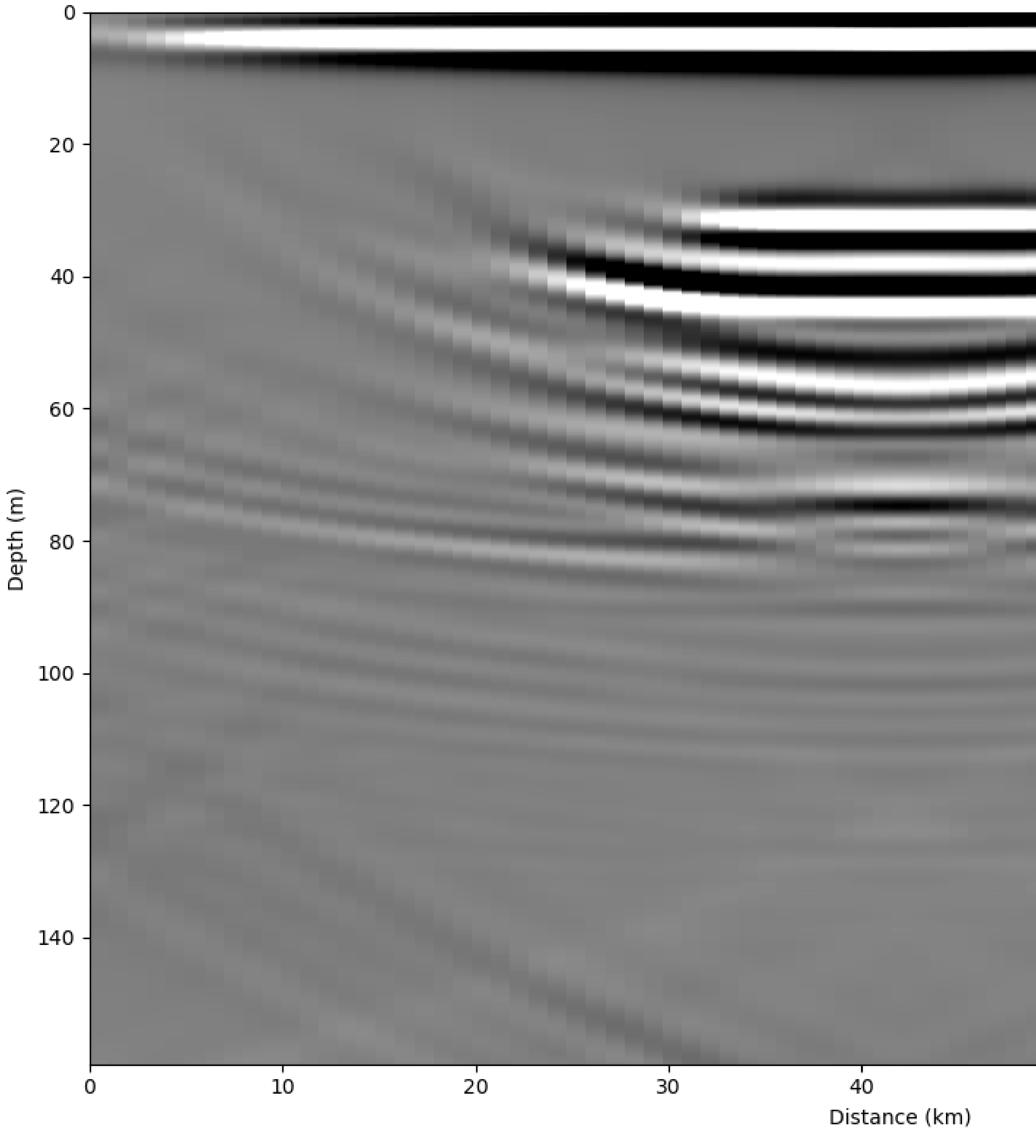
Phase-Shift Migration

The second frequency-wavenumber migration routines is actually a set of a few called phase-shift migration (sometimes Gazdag migration). A phase-shifting operator $e\hat{}-ik_z z$ is applied at each z-step in downward continuation. These methods are advantageous in that they allow variable velocity as one steps down. Generally, this only allows vertical velocity variation but there is also a case which accomadates lateral velocity variation (phase-shift plus interpolation).

```
impdar migrate --mtype phsh synthetic.mat
```

Constant velocity phase-shift migration is the default in ImpDAR, so it can also be called as,

```
impdar migrate synthetic.mat
```



Much like the result from Kirchhoff migration, we see upward dipping 'smileys' in this migrated image.

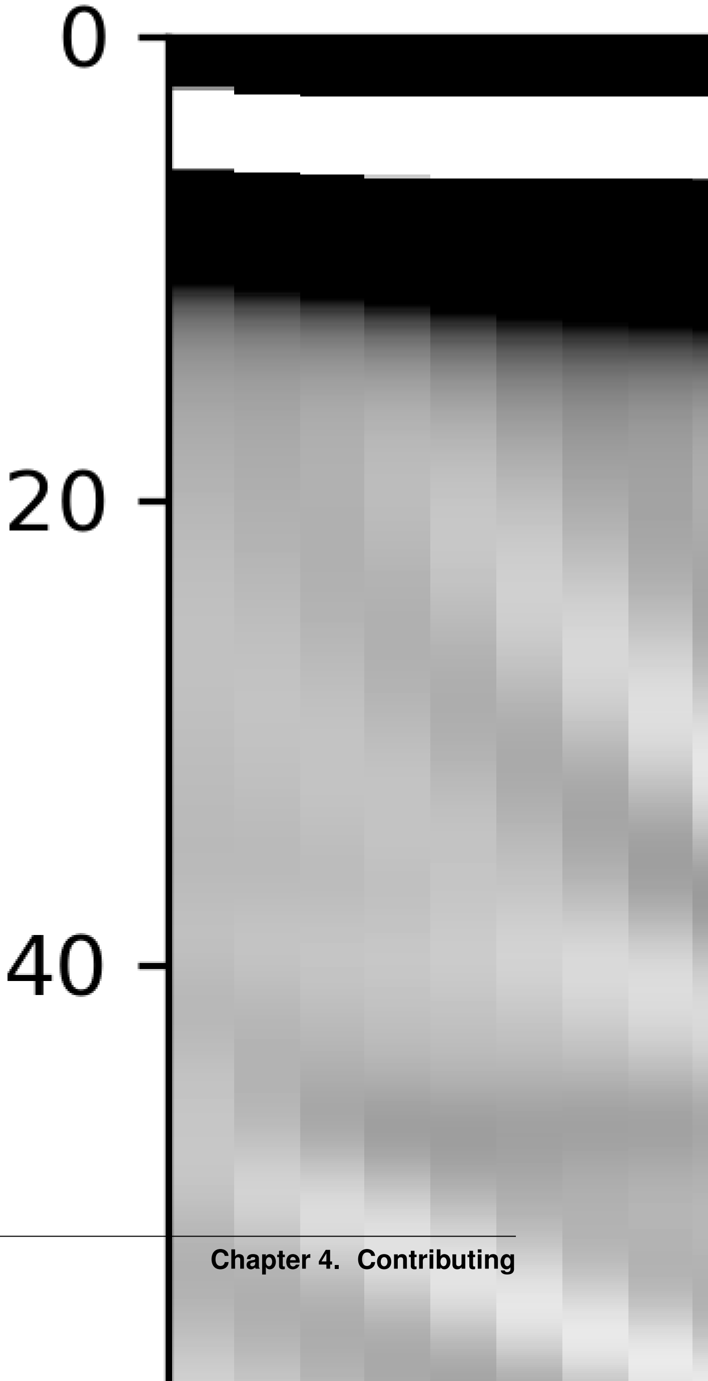
Summary of Phase-Shift Migration:

- Strengths – Accomodates velocity variation (particularly appropriate for vertical variations, i.e. in snow/firn or similar).
- Weaknesses – Maximum dip angle.

SeisUnix Migration Routines

There are many migration routines implemented in the seismic processing package, SeisUnix. With ImpDAR, we have no intent to replicate the work that they have done; instead, we allow the user to easily convert radar data to .segy, migrate with SeisUnix, then convert back, all in a kind of black-box fashion with only one command. If SeisUnix is not installed, this command with raise an error.

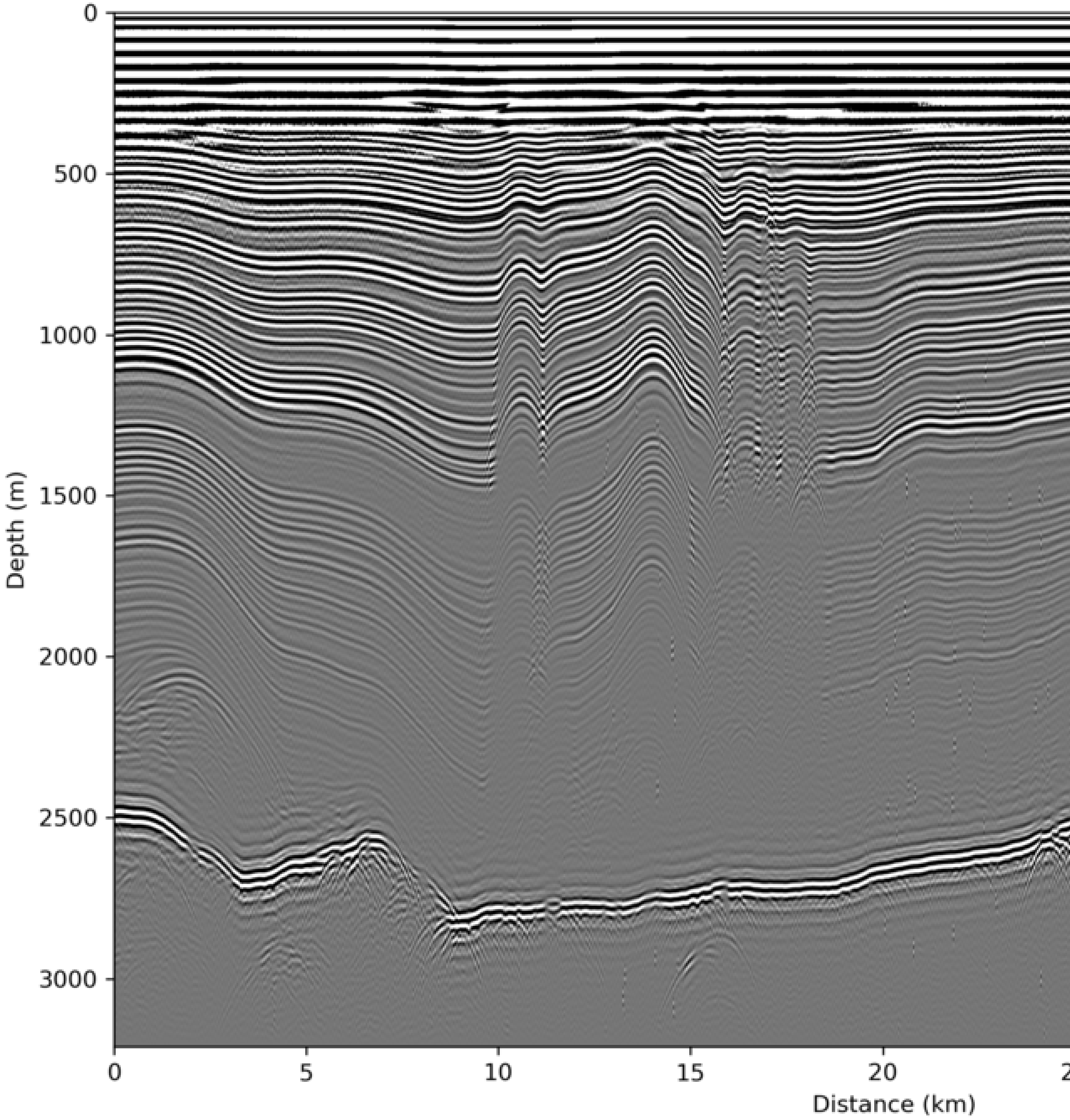
```
impdar migrate --mtype sumigtk synthetic.mat
```


0
20
40

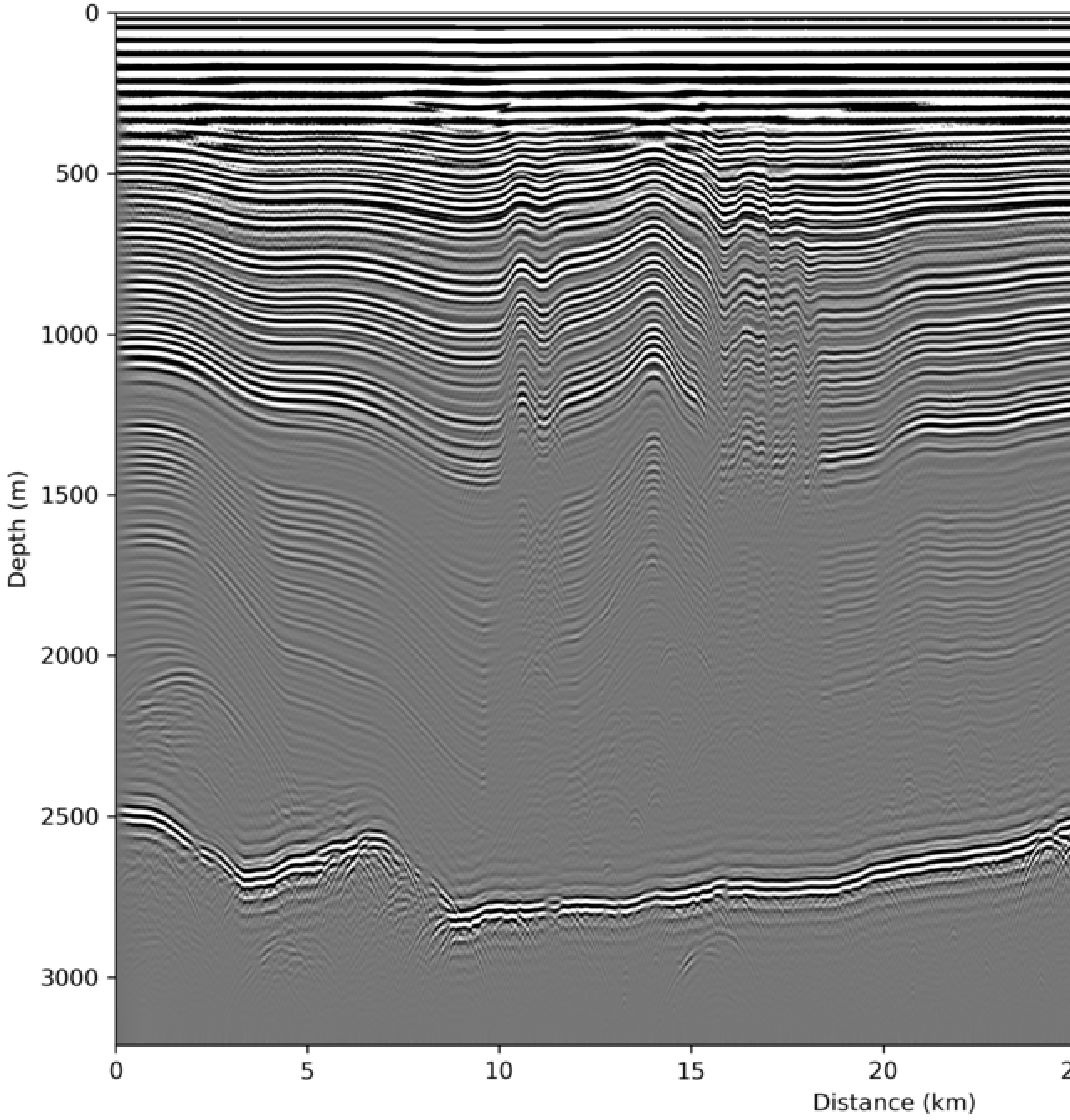
Data Example

Below is a real example of migration in ImpDAR for 3-MHz ground-based data from the Northeast Greenland Ice Stream (Christianson et al., 2014).

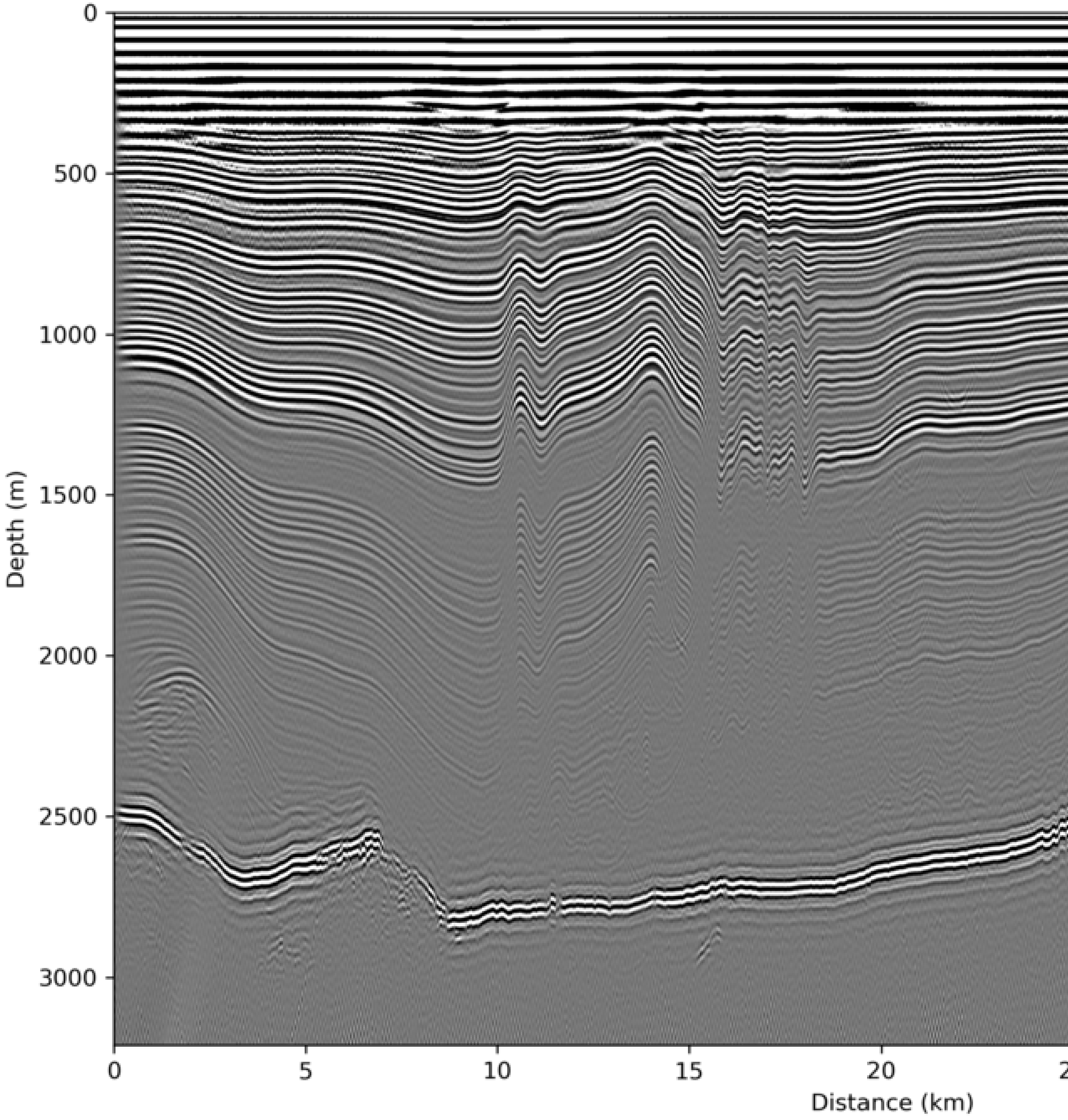
Unmigrated Data:



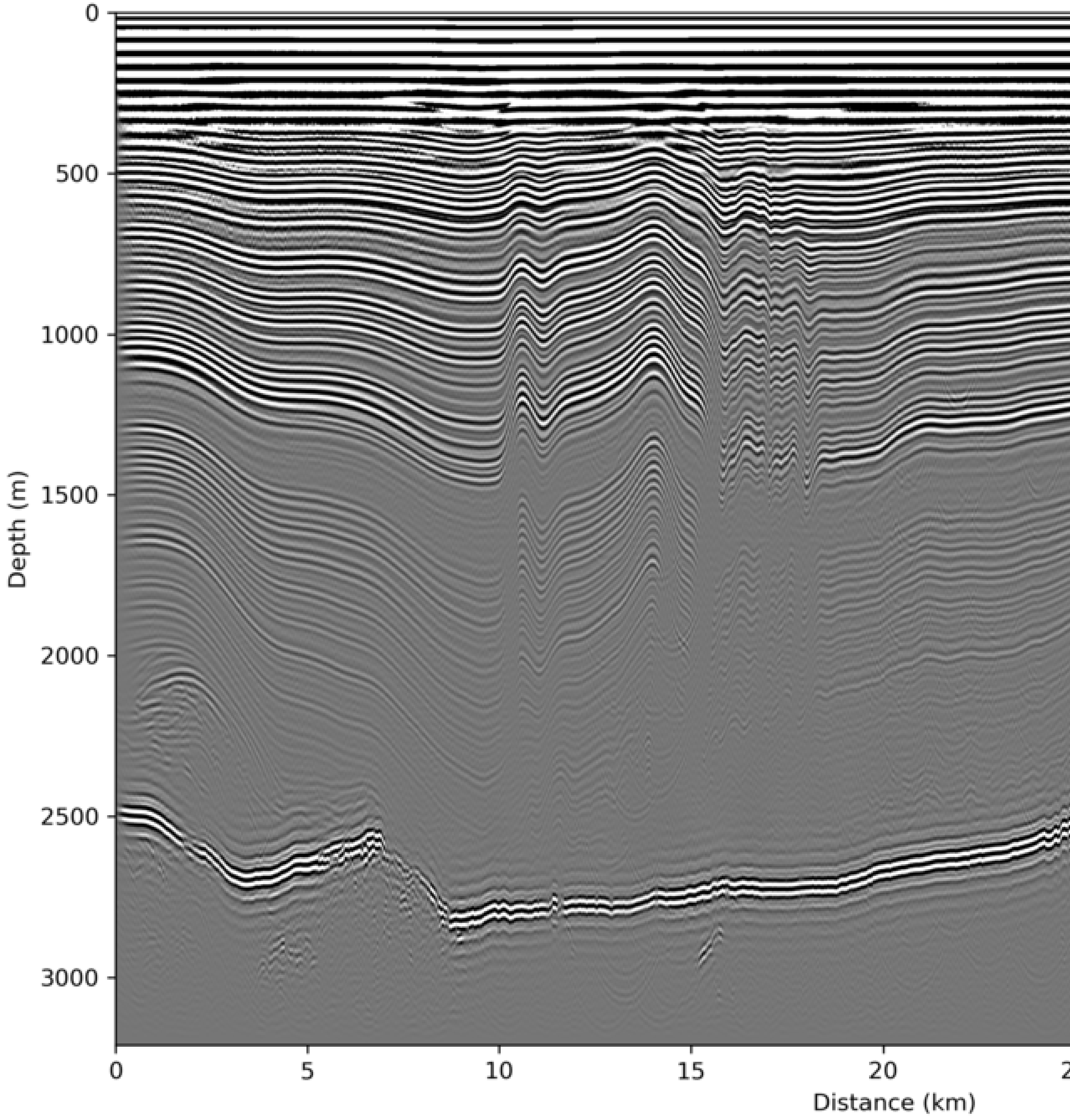
Stolt:



Phase-Shift:



SeisUnix T-K:



References:

Yilmaz (2001). Seismic Data Processing.

Sherrif and Geldart (1995). Exploration Seismology.

Hagedorn (1954). Seismic Imaging Migration.

Stolt (1978). Migration by Fourier Transform. *Geophysics*

Gazdag (1978). Wave Equation Migration with the Phase-Shift Method. *Geophysics*

Christianson et al. (2014). Dilatant till facilitates ice-stream flow in northeast Greenland. *Earth and Planetary Research Letters.*

Plotting examples

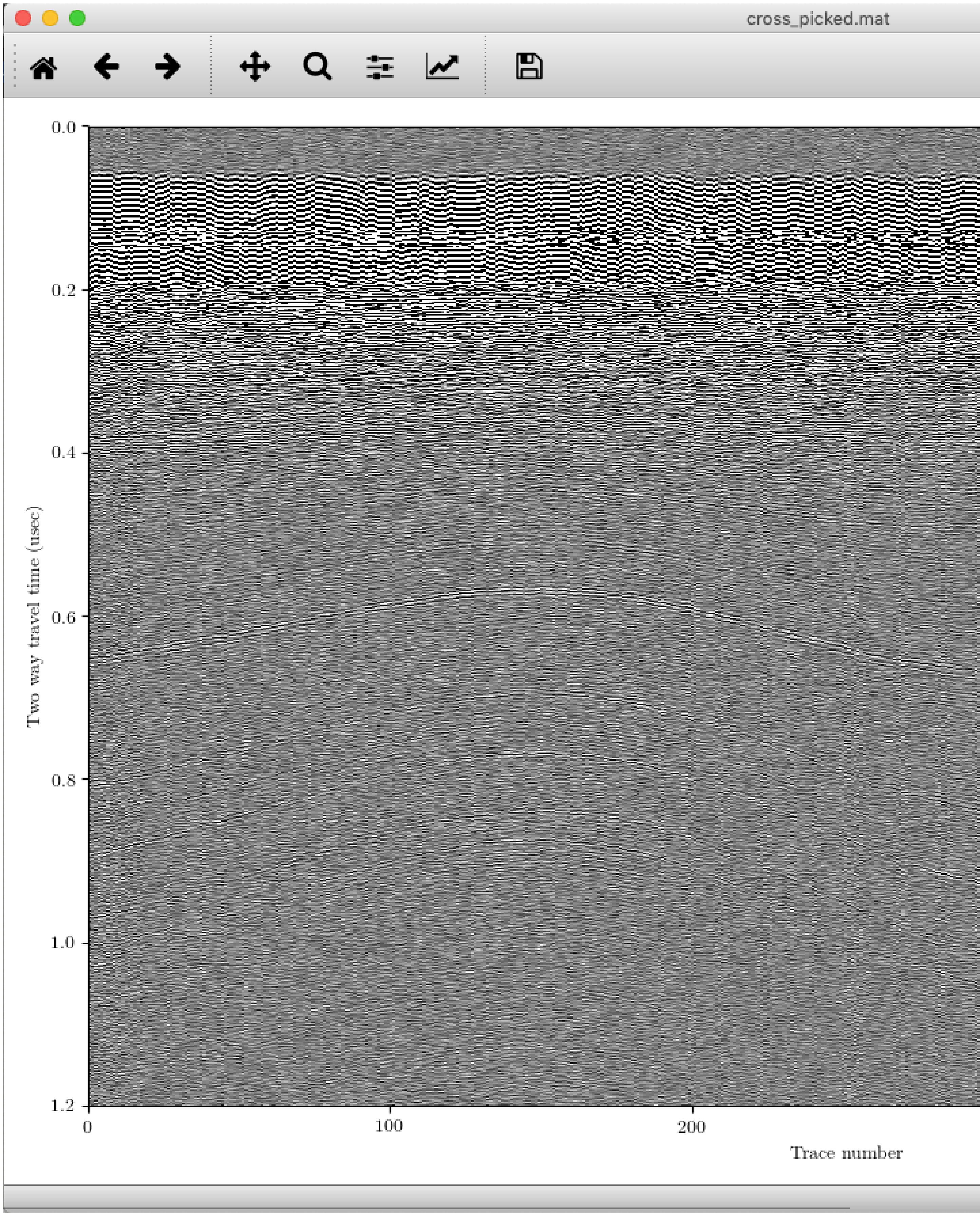
Visualizing output is an essential piece of processing and the disseminating radar data. You likely will need to look at the output many times so that you can discern the effect of different processing steps, and then you will likely want to make a figure at the end. With these two use cases in mind, ImpDAR provides both command-line plotting, for quick and easy visualization, and API calls more customized plots.

impplot

ImpDAR permits you to make plots by calling `impdar plot [fns]` with a few options, but I recommend using `impplot` instead. The syntax is cleaner and it is more clear what you are doing. There are different types of plots you can make with `impplot`, described below, but it is first worth noting you can always add the `-s` directive to save the output to a file rather than pulling it up in a matplotlib figure window.

radargram

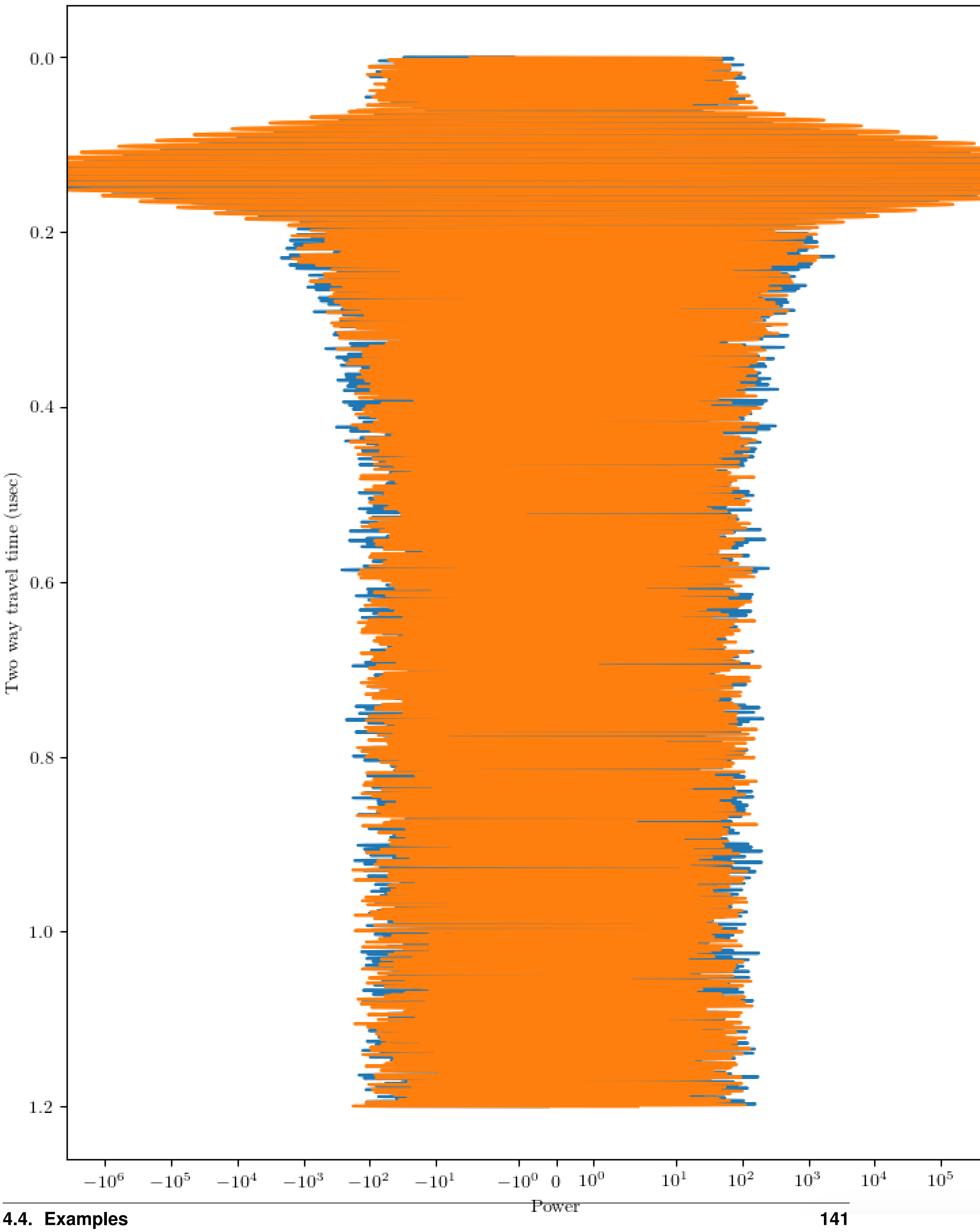
The most common thing to plot is probably the full radargram. The basic syntax is `impplot rg [fns]`, with additional options described with *impplot*. When you run `impplot rg` you will get something like this popping up in an interactive window.


cross_picked.mat
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Two way travel time (usec)
0
100
200
Trace number

You can pan and zoom around the plot, and determine what other processing steps you might want to take. You can this with multiple filenames and get a group of plots. If there are picks in the file, these will be displayed as well, though you can deactivate this feature with `-nopicks`.

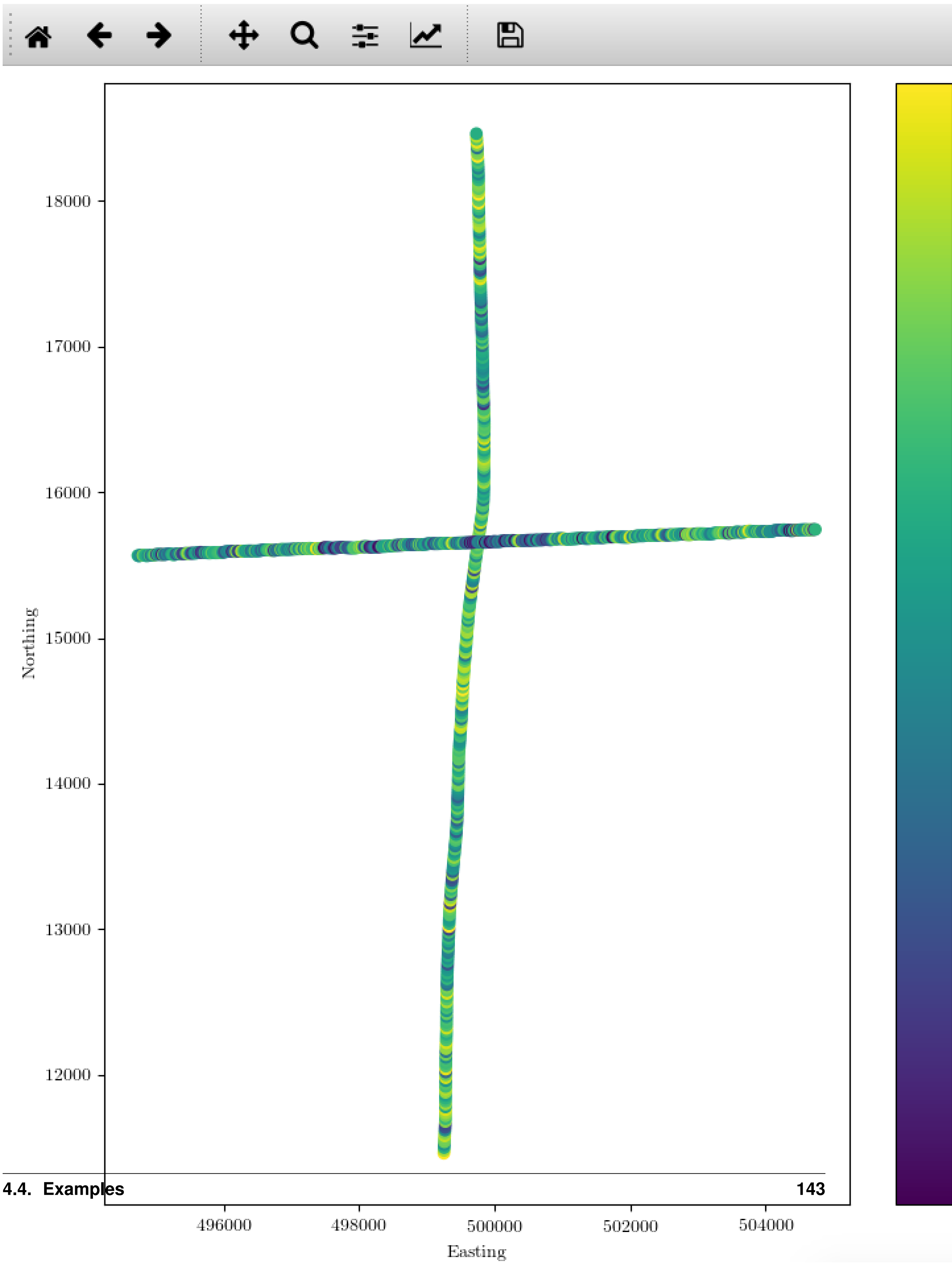
traces

Sometimes you may want to look at how the samples in an individual trace, or group of traces, vary with depth. A range of traces can be plotted with `impplot [fns] trace_start trace_end`. The output is something like this.


Two way travel time (usec)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Power
-10^6
-10^5
-10^4
-10^3
-10^2
-10^1
-10^0
0
10^0
10^1
10^2
10^3
10^4
10^5

power

This command is used to look at the variability in reflected power in space. You will get a single plot with the return power of a given layer in all the different profiles called. The syntax is `impplot [fns] layer`. If there are projected coordinates, those will be used, and otherwise you are stuck with lat/lon. The result for two crossing profiles might look something like this.

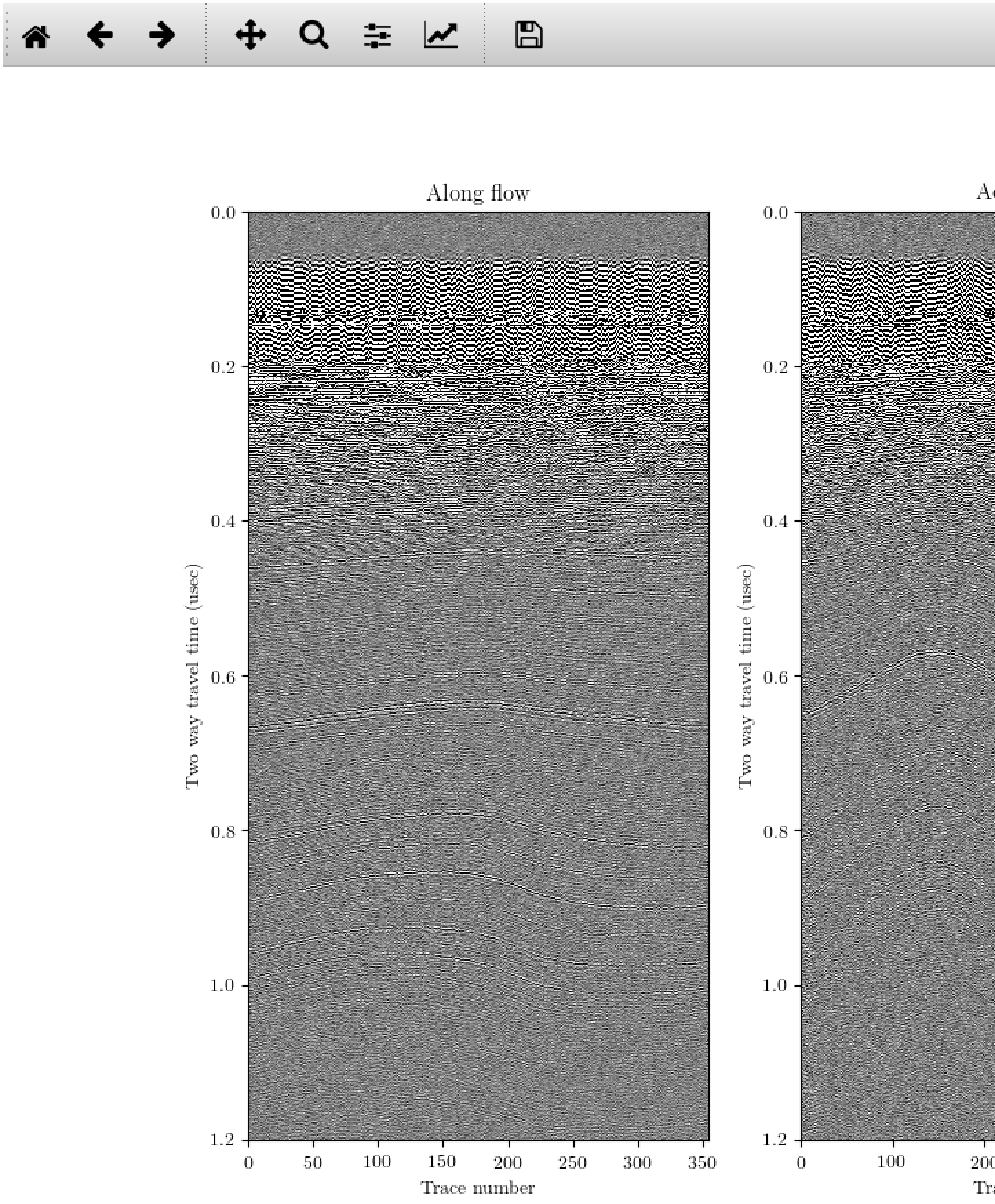

18000
17000
16000
15000
14000
13000
12000
Northing
496000
498000
500000
502000
504000
Easting

API

There are several reasons you might want to use an API call rather than `impplot`: perhaps want to modify the output, perhaps by annotating it or plotting on top of it; you may want to put a panel made by ImpDAR in a figure with other subplots; or maybe you just need to have multiple panels plotted by ImpDAR. Regardless, in these cases I would recommend loading up the data then using the explicit plotting functions in *the plotting library*. I'll just give an example of a several-panel plot with all panels produced through ImpDAR. Say you want to make a 3-panel plot showing two profiles and the power returned from both. You could use

```python
import matplotlib.pyplot as plt
from impdar.lib import RadarData, plot

# Load the data we are using; in this case they are already processed
profile_1 = RadarData.RadarData('along_picked.mat')
profile_2 = RadarData.RadarData('cross_picked.mat')

# Make the figure we will plot upon--need some space between axes
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, gridspec_kw={'hspace': 1.0}, figsize=(12,␣
↪8))

# plot the two radargrams
plot.plot_radargram(profile_1, fig=fig, ax=ax1)
plot.plot_radargram(profile_2, fig=fig, ax=ax2)

# Now look at their return power in space on layer 5
plot.plot_power([profile_1, profile_2], 5, fig=fig, ax=ax3)

#document what we are looking at
ax1.set_title('Along flow')
ax2.set_title('Across flow')
ax3.set_title('Layer 5\nreturn power')


# see how it turned out
plt.show(fig)
```

And this will produce a nice 3-panel figure (though we would certainly want to do a better job with reasonable aspect ratios for most applications).


Along flow
Ac
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Two way travel time (usec)
0
50
100
150
200
250
300
350
Trace number
0
100
200
Tra

Picking examples

It is hard to give examples of GUI use, but the different functionalities are fairly well documented along with *imppick*. In particular, look through the workflow on that page for a normal procedure for interpreting a profile.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index